Debreceni Egyetem

Informatika Kar

Mobil banki alkalmazás Android platformon

Belső témavezető:	***Külső témavezető:***	***Készítette:***
Dr. Fazekas Gábor	*Gyenes Gábor*	*Száldobágyi Lóránd Lajos*
Egyetemi docens	*IND Kft. Termékvezető*	*Programtervező matematikus*

Debrecen

2009

Tartalomjegyzék

1 Bevezetés

Az elmúlt évtizedben a vezeték nélküli iparág jelentős fejlődésnek indult, ami nagyrészt a mobiltelefonok elterjedésének köszönhető. Különböző technológiák jelentek meg, mint a 3G, a Wifi és a Bluetooth; miközben a hardware is továbbfejlődött és egyre kisebb lett.

Napjaink mobil eszközeiben – legyen az egy PDA, smartphone, vagy egy átlagos mobil telefon - a mobil Internet elérés már alap funkciónak számít. Egyetlen mobilkészüléken már szinte az összes szolgáltatás elérhető. Olvasható rajtuk e-mail, böngészőn keresztül elérhető a web, megjelentek az IM (instant messenger) típusú szoftverek és egyéb internet alapú alkalmazások, miközben a 3G szélessávú technológia még további újabbak megjelenésének ad lehetőséget. Mindezek következtében szolgáltatásokból és szoftverekből is egyre több jelenik meg a piacon. Operációs rendszerekből is széles a kínálat.

Az Internet banki szolgáltatásokat nyújtó bankok számára a mobil- és webes technológiák konvergenciája megfelelő lehetőséget teremt arra, hogy elektronikus szolgáltatásait közvetlenül tegye elérhetővé mobil előfizetők ezrei számára. Mindezt aránylag könnyen, a mobil operátortól függetlenül.

A diplomamunka témát adó cég, az IND Kft. is felfigyelt a banki szférában alkalmazott mobil technológiákban rejlő lehetőségekre, révén hogy a vállalat fő profilja banki alkalmazások készítése, amely területen számos kitűnő referenciát szerzett mind hazai, mind külföldi viszonylatban. A cég Internet banki termékeit többek között a platformfüggetlenséget biztosító Java és XML technológiákra építi, amelyek eszközt biztosítanak az egyszerű rendszerintegrációra.

Az IND számos mobil technológiával dolgozott már: WAP, SMS, J2ME és az új mobilos XHTML. De folyamatosan keresi az új lehetőségeket és szabványos megoldásokat.

Jelen dolgozatom célja bemutatni napjaink mobil eszközeit, ezek elterjedését, típusait, előnyeit és hátrányait, továbbá a mobil bankolás bemutatása, leggyakrabban használt mobil banki szolgáltatások és osztályozásuk, valamint a mobil bankolásra alkalmas technológiák ismertetése.

Célom a mobil bankolásra alkalmas technológiák közül a mobil kliens alkalmazásra egy konkrét alkalmazás megvalósítása Android platform alatt és az Android bemutatása.

2 Mobil eszközök

2.1 Mi is az a mobil eszköz

Mobil eszköz, olyan számítástechnikai eszköz, amely helytől függetlenül nyújt a felhasználónak informatikai szolgáltatásokat.

2.2 Kialakulásuk

A mobil eszközök fogalma a mobil telefonok kialakulásával kezdődött.

2.2.1. Korai évek

1947 decemberében Douglas H. Ring és W. Rae Young, Bell Labs mérnökök, hexagonal cellákat terveztek mobil telefonokhoz. Philip T. Porter, szintén a Bell Labs-nál, azt javasolta, hogy a cella tornyok inkább a hexagonok sarkaiban legyenek mintsem a közepén és legyen közvetlen antenna, ami 3 irányban továbbíthat/fogadhat 3 szomszédos hexagon cellába. A technológia akkor még nem létezett és a frekvenciák sem voltak elosztva. A cellás technológia elektronikáját az 1960-as években Richard H. Frenkiel és Joel S. Engel a Bell Labs-nál fejlesztették ki.

Európában a rádiótelefonokat először az első osztályú személyvonatokon használták Berlin és Hamburg között 1926-ban. Ezzel egy időben vezették be a rádiótelefonálást a személyszállító repülőgépeken a légiközlekedés biztonságáért. Később széles körben alkalmazták a német tankokban a második világháború alatt. Mindezen esetekben az alkalmazás korlátozva volt a szakemberekre, akiket kiképeztek a berendezés használatára. Az 1950-es évek elején a rajnai hajók az elsők között voltak, akik képzetlen felhasználóként alkalmazták a rádiótelefonálást.

Az első valódi mobil telefonok az 1950-es évektől léteznek.

Az első teljesen automatikus mobil telefon rendszert – amit MTA-nak (Mobile Telephone system A) hívnak – az Ericsson fejlesztette ki és 1956-ban hozták kereskedelmi forgalomba Svédországban. Ez volt az első rendszer, amely nem kívánt semmilyen kézi irányítást, de hátránya a telefon 40 kg-os súlya volt. Az MTB-t, ami egy továbbfejlesztett verzió tranzisztorokkal, és a súlya 9kg, 1965-ben mutatták be és DTMF (Dual Tone Multiple Frequency) jeladásra használták.

1967-ben minden mobil telefonnak egy alap állomás által szolgáltatott cella területén belül kellett maradni a telefonhívás alatt. Ez nem szolgáltatta az automatikus telefon szolgáltatás folyamatosságát, a mobil telefonok mozgását számos cella területen keresztül. 1970-ben Amos E. Joel, Jr., egy másik Bell Labs mérnök, felfedezett egy automatikus "call handoff" rendszert amely lehetővé teszi a mobil telefonok mozgását egy egyszerű párbeszéd alatt, anélkül hogy megszakadna.

1971 decemberében az AT&T előterjesztett egy javaslatot a Federal Communications Commission (FCC)-hez cellás szolgáltatásra. Évekkel a meghallgatás után 1982-ben az FCC elfogadta a javaslatot az Advanced Mobile Phone Service (AMPS) -re és kiosztotta a frekvenciákat a 824-894 MHz sávban. Az analóg AMPS-t 1990-ben kiszorította a Digital AMPS.

Az első igazi sikeres nyilvános kereskedelmi mobil telefon hálózatok közül az egyik az ARP volt Finnországban, amit 1971-ben vezettek be. Utólag az ARP-t néha zéró generációs (0G) cellás hálózatnak tekintik, alig fölötte lévén a korábbi szabadalmazott és határolt közvetítő hálózatoknak.

Dr. Martin Cooper a Motorolától 1973-ban kezdeményezte az első analóg mobil telefon hívást egy nagyobb prototípuson az Egyesült Államokban.

1978-ban, a Bell Labs elindította az első kereskedelmi cella hálózat próbáját Chicagoban AMPS-t használva.

2.2.2. Első generáció

A cellás távközlés első kereskedelmi bevezetése 1979-ben volt Tokióban Japánban a NET által. 1981-ben a NMT rendszert bevezették Dániában, Finnországban, Norvégiában és Svédországban. Az első kézben tartható mobil telefon az amerikai piacon a Motorola_Dyna 8000X volt 1983-ban. A mobil telefonok elszaporodtak a sokszoros alap állomással rendelkező cellás hálózattal, melyek viszonylag közel helyezkedtek el egymáshoz. Ekkor analóg átvitel volt minden rendszerben. Hamarosan a testes egységeket kiváltották a szállítható, pénztárca méretű telefonokra. Ezek a rendszerek (NIT, AMPS, SACS, RT MI, C-Net, és Radio com 2000) később első generációs mobil telefonokként váltak ismertté.

2.2.3. Második generáció

Az 1990-es években kezdték bevezetni a második generációs mobil telefon rendszereket, mint a GSM, IS-136 ("TDMA"), iDEN és IS-95 ("CDMA"). Az első kereskedelmi digitális cellás telefonhívást 1990-ben az USA-ban kezdeményezték, 1991-ben Finnországban megnyitották az első GSM hálózatot. A 2G telefon rendszereket digitális körök által kapcsolt átvitel jellemezte. Általában a 2G rendszereknél használt frekvencia Európában kissé magasabb, például a 900 MHz frekvenciát használták mind az 1G mind a 2G rendszernél és így az 1G rendszereket gyorsan megszüntették, hogy ne zavarják a 2G rendszereket.

A második generáció bemutatta az új kommunikációs lehetőséget, az SMS szöveges üzenetet, kezdetben a GSM hálózatokon, végül minden digitális hálózaton.

Szintén a 2G mutatta be a média felhasználás lehetőségét a mobil telefonokon, mikor Finnországban bemutatták a letölthető csengőhangokat.

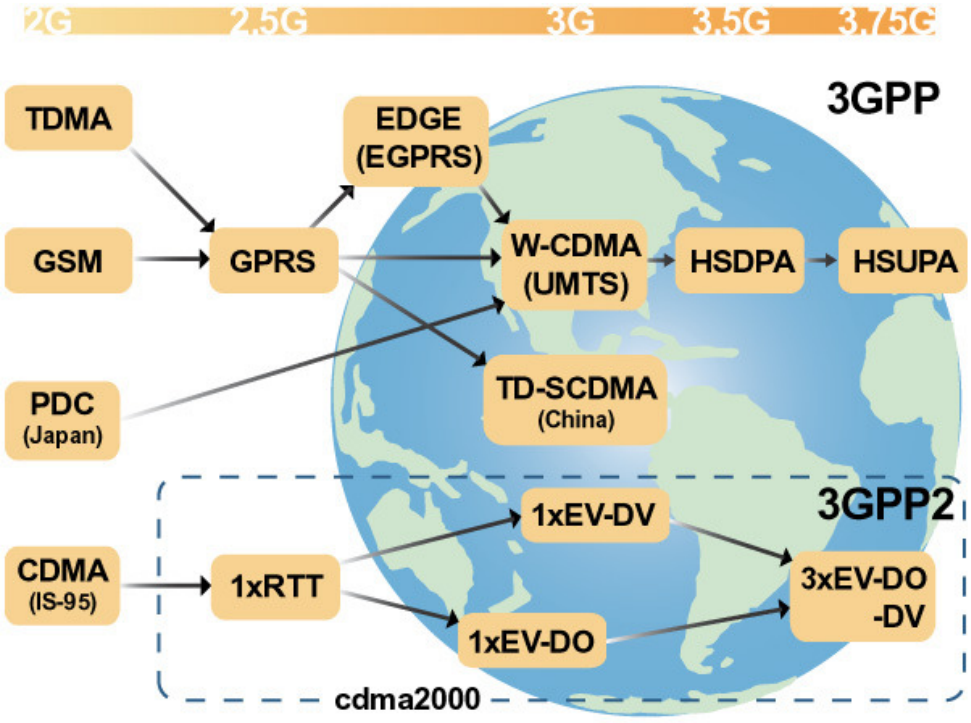
2.2.4. Harmadik generáció

Nem sokkal a 2G hálózatok bemutatása után elkezdték fejleszteni a 3G rendszereket. Elkerülhetetlenül több különböző szabvány volt különböző versenyzőkkel, akik a saját technológiájukat reklámozták. Eléggé különbözik a 2G rendszertől, habár a 3G jelentést szabványosították az IMT-2000 szabványosító eljárásban. Ez az eljárás nem a technológiát szabványosítja, inkább a követelményeket (2 Mbit/s maximum bejövő adat sebesség, 384 kbit/s kimenő).

Az első 3G kereskedelmi hálózatot Japánban vezette be az NTT DoCoMo 2001-ben a WCDMA technológiát alkalmazva. 2002-ben bevezette az első 3G hálózatokat a rivális CDMA2000 1xEV-DO technológián alapulva az SK Telecom és KTF Dél-Koreában, és a Monet az USA-ban. 2002 végére a második WCDMA hálózatot indította el Japánban a Vodafone KK (most Softbank). Márciusban az első európai 3G bevezetések Olaszországban és az Egyesült Királyságban voltak WCDMA-n. 2003-ban további 8 kereskedelmi bevezetés volt, hatszor a WCDMA-n és kétszer az EV-DO szabványon.

A 3G rendszer fejlődése alatt 2.5G rendszerek, mint CDMA2000 1x és GPRS, fejlődtek ki, a létező 2G hálózatok kiterjesztéseként. Ezek rendelkeznek a 3G néhány tulajdonságával anélkül, hogy teljesítené a kívánt magas adatátviteli sebességet vagy a multimédia szolgáltatások teljes körét. A CDMA2000-1X elméleti maximum adatátviteli sebessége több mint 307 kbit/s. Ezek után jön az EDGE rendszer, amely elméletileg lefedi a 3G rendszer szembeni követelményeket.

2007 végére 295 millió előfizető volt a 3G hálózatokon világszerte. Körülbelül kétharmaduk a WCDMA szabványon és egyharmaduk az EV-DO szabványon. [1]



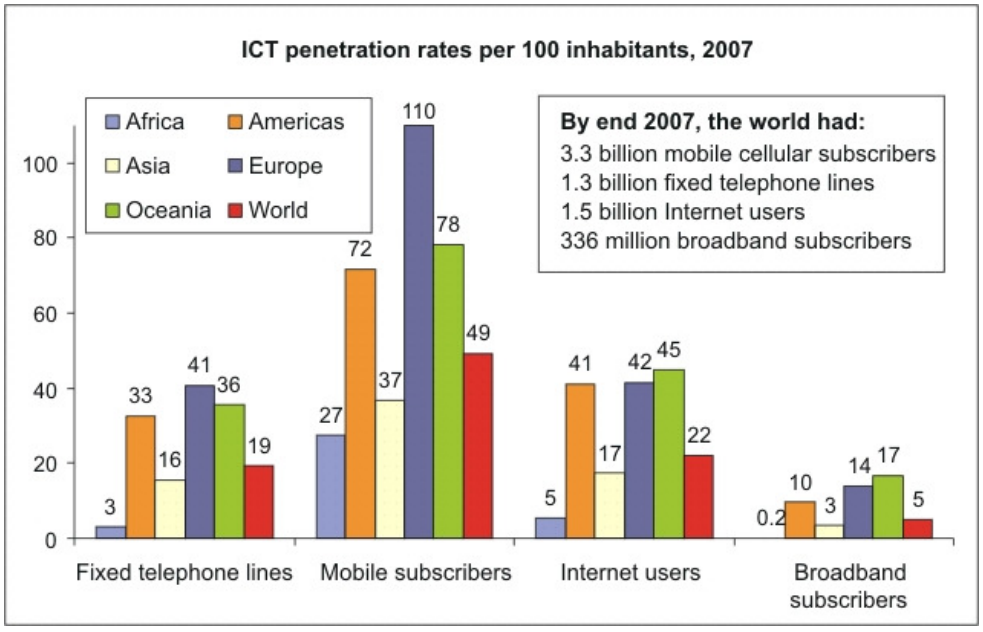
1. ábra: Mobil hálózati szabványok fejlődése

2.3 Elterjedése

A mobil eszközök, köztük a mobil telefonok, korunk egyik legnagyobb technikai vívmányai közé tartoznak. A marok telefonokba integrált számtalan funkciónak, és az ezek nyújtotta

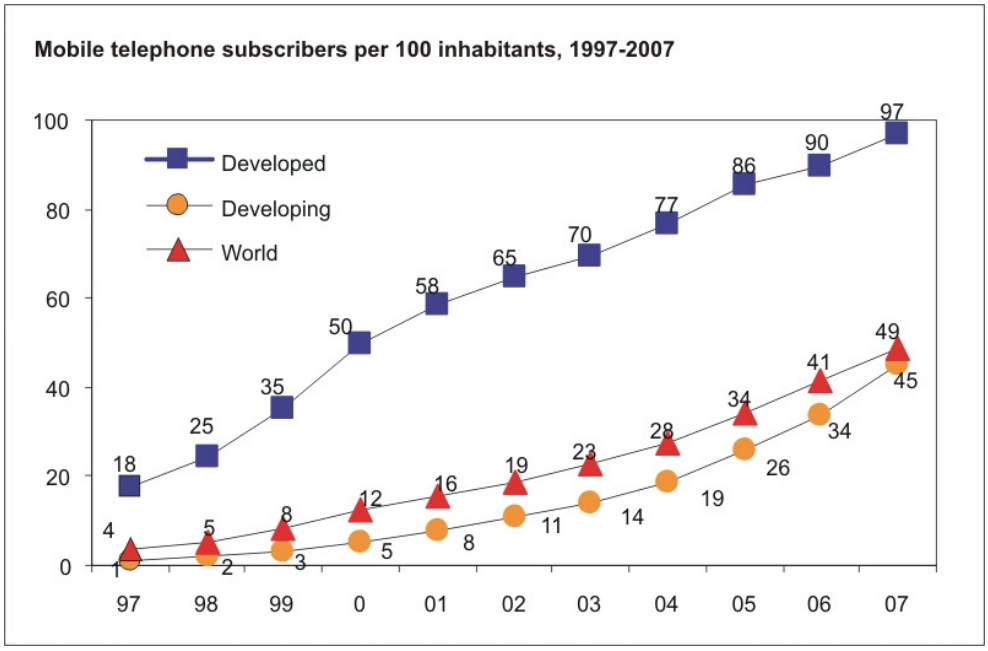
kényelemnek köszönhetően a mobiltelefonok mára széles körben elterjedt eszközei a kommunikációnak, és az adatalapú szolgáltatások elérésének.

A következő rész némi betekintést nyújt a naprakész tényekbe az ICT-ről (Information and Communication Technology) világszerte. 2007 végére majdnem minden második embernek van mobil telefonja. Európában a mobil telefonok térhódítása meghaladja a 100%-ot. Afrikában minden negyedik, Ázsiában minden harmadik embernek van mobil telefonja. A magas szintű verseny és árakban történt csökkenés tette lehetővé a digitális szakadék csökkenését a mobil telefonokban. A digitális szakadék marad a fő probléma az internettel kapcsolatban és különösen a szélessáv felvétellel.



2. ábra: ICT terjedési arány

Ahogy az ábrán látható, a mobil növekedés a fejlődő világban marad a legerősebb. 2007 végére a fejlődő világban 100 lakosból 45 rendelkezett mobil telefonnal. [2]



3. ábra: Mobil telefon előfizetők száma 100 lakosra

Az NHH (Nemzeti Hírközlési Hatóság) adatai szerint Magyarországon 2008 szeptemberében a mobil-előfizetések száma 83 ezerrel nőtt, így a hónap végén a három szolgáltató ügyfeleinek száma összesen 11 millió 771 ezer volt és 100 lakosra 117,2 előfizetés jutott az egy évvel korábbi 104,4 után. 2007 szeptembere óta a magyar piacon az ügyfélszám 1,273 millióval nőtt, ezen belül 2008 első 9 hónapjában a növekedés 741 ezeres volt.

A forgalmazásban részt vevő kártyák száma 2008. szeptember végén 10 millió 490 ezer volt, 68 ezerrel több, mint augusztus végén. A forgalmazásban részt vevő kártyák száma az idei év első 9 hónapjában 360 ezerrel, 2007. szeptember vége óta pedig 834 ezerrel nőtt. [3]

2008. szeptember

Előfizetések száma	11 771 197	A hívásfogadásra képes aktív SIM kártyák száma.
Forgalmazásban részt vevő előfizetések száma	10 489 696	Az utolsó három hónapban forgalmazó aktív SIM kártyák száma.
100 lakosra jutó előfizetések száma	117,2	A hívásfogadásra képes aktív SIM kártyák száma viszonyítva a KSH által publikált legfrissebb népességszámhoz
Előfizetések számának változása	100,71%	Előző hónaphoz képest.
	113,33%	Előző év hasonló időszakához képest.

2.4 Típusai

A mobil eszközöknek hat fő típusa van: laptop-ok, notebook-ok, tablet PC-ék, PDA-k, smartphone-ok, és mozgatható adat terminálok. Az első hármat gyakran nevezik "hordozható" és a második hármat "kézben hordozható" számítógépnek.

2.4.1. Laptop

Laptop - néha "asztali számítógépet helyettesítő PC"-nek nevezik – az asztali számítógép teljesítményét nyújtja mobil környezetben. Észak-Amerikában széles körben elérhetőek 2002 óta. A tipikus laptop képernyője 14 - 17inch, rendelkezik belső DVD-ROM vagy DVD-RW meghajtóval, széles billentyűzettel, integrált modemmel, hálózattal, Bluetooth-al, és Wi-Fi

képességgel, magas minőségű integrált audio és beszélő rendszerrel, három vagy több óra akkumulátoridővel, alacsony akkumulátor fogyasztással és a fejlesztés képességével. A teljesítményük ellenére csak 3-12 font a súlyuk. Számos laptopnak nem csak beépített billentyűzete van, de szintén van touchpad vagy pontozó pálca a bevitelre. Általában egeret is lehet hozzácsatlakoztatni. Mindezen előnyök ellenére, az asztali számítógépek továbbra is nagyon népszerűek, mert sokkal tartósabbak és kevésbé költséges.

2.4.2. Notebook

A Notebook könnyűsúlyú számítógépek, melyeknek nagyon vékony profilja van, 12 -14 inch képernyő, nincs belső DVD vagy CD rendszer, nincs belső floppy meghajtó, korlátozott grafikus képességek, integrált modem és hálózati csatlakozás, kis billentyűzet, alacsony energia fogyasztás és négy vagy több óra akkumulátoridő. A fejleszthetőség is általában korlátozott. A tipikus notebook súlya kb. 1.5 -3 font. Míg néha nem különböznek a laptop computertől, az előző nagyobb, erőteljesebb és sokkal közelebb áll az asztali számítógéphez.

2.4.3. Tablet PC-k

Tablet PC palatábla alakú mobil számítógépek érintő képernyővel vagy grafikus tablet/képernyő hybriddel felszerelve. Így a felhasználó egy tűvel, digitális tollal vagy ujjbeggyel irányíthatja a számítógépet billentyűzet vagy egér helyett. Habár csatlakoztatható külső billentyűzet vagy egér. Néhány tablet PC szintén támogatja a hang felismerést. A képernyő általában kicsi, gyakran 8 inch-nél is kisebb. A tabletek processzor sebessége és memóriája fele a notebook számítógépeknek, de a megoldás összehasonlítható és a súlya is kevesebb, mint a fele a notebook számítógépeknek.

2.4.4. PDA-k

Míg a PDA-k (Personal Digital Assistants) a legkeresettebb kézben hordozható mobil számítógép berendezések között vannak 2000 óta, kezdetben elsődlegesen az ütemterv nyomonkövetésére, nevek és címek jegyzékének fenntartására, és e-mail fogadásra használták. Csak 2003 óta növekedett a processzor sebesség és a memória addig a pontig, ahol praktikus egy Web böngésző, cellás telefon és fax képesség. Ez kiküszöbölte, hogy szükség legyen egy külön cellás

telefon hordozására. A cellás telefongyártók a “smartphone”-okkal válaszoltak, melyek tartalmaznak Web böngészőt.

A PDA-k tipikus tulajdonságai az érintő képernyő az adatbevitelre, egy memória kártya nyílás az adattárolásra, és Bluetooth és/vagy Wi-Fi támogatás, az adatok bevitele általában virtuális billentyűzeten történik, ahol a billentyűzet az érintő képernyőn látható, így a bevitel a betűk begépelésével ujjal vagy tűvel történhet. A RIM népszerű BlackBerry PDA-jának nem csak érintő képernyője van, de egy teljes billentyűzet és görgők az adatbevitel és a navigálás megkönnyítésére. A Palm Treo szintén PDA. A “Pocket PC” egy a megadott PDA-k közül, amelyik a Microsoft Windows mobil operációs rendszert használja.

2.4.5. Smartphones

A Smartphone kompromisszum a PDA és egy cellás telefon között, a cellás telefon részre összpontosítva. Egy smartphonera programokat installálhat a felhasználó, információkat tárolhat, e-maileket fogadhat, de a Web hozzáférés korlátozott. Ezeknek van operációs rendszerük. Használt operációs rendszerek: Symbian, Windows Mobile, Rim BlackBerry, iPhone OS, Linux, Palm OS, Android.

Nincs ipari szabvány, ami definiálná mi is az a smartphone, ennek következtében néhány mobil készüléket, amelynek több képessége van, mint egy alap cellás telefonnak smartphoneként kínálhatják eladásra. Az Apple iPhone és a Google 3G-nek annyi PDA-hoz hasonló tulajdonsága van, hogy néhány bíráló PDA-ként sorolta be őket.

2.4.6. Mozgatható adat terminálok

A mozgatható adat terminálok olyan készülékek, melyek az adatok beírására és visszakeresésére használhatók vezeték nélküli átvitelen keresztül. Az adatbevitel történhet billentyűzettel vagy vonalkód olvasóval. A fő alkalmazásuk a készletkönyvelésben van. A mozgatható adat terminálok gyakran vezeték nélkül management szoftverrel ellátva futnak, melyek megengedik az adatbázissal való kapcsolatot egy szerveren.

2.5 Előnyei/Hátrányai

2.5.1. Előnyei

- Mindig rendelkezésre állnak, bárhol használhatóak
- Szinte mindig bekapcsolt állapotban tarthatóak, könnyen és gyorsan elérhető funkciók
- Személyes
- Sokoldalú
- Kisméretűek és kis tömegűek, így könnyen magunkkal hordhatjuk

2.5.2. Hátrányai

- Kis kapacitású CPU (300MHz—1-2GHz)
- Kis méretű memória
- Kis méretű képernyő

  Mobil képernyő méretek: 96x65, 128x128, 128x160, 176x208/220, 240x320, 320x480, 640x200/360/480, 800x352/400/480

  Uralkodó képernyő méret a 240x320-as.
- Korlátozott adatbeviteli lehetőségek (billentyűzet, trackball/joystick, toll, érintés, mikrofon, kamera, GPS)
- Erősen korlátozott energiafogyasztás
- Biztonsági hiányosságok:elvesztés, eltulajdonítás, tűzfal, vírusvédelem, vezeték nélküli megoldások
- Különböző operációs rendszer (különböző programozási nyelvek)

Manapság a mobil sávszélesség nem számít hátránynak, mivel már 14,4 Mbit/mp letöltési sebesség is elérhető.

3 Mobil szolgáltatás

A mobil eszközök felhasználási területe igen kiterjedt, alkalmasak mind személyes, mind vállalati felhasználásra. Egy vállalat az IT infrastruktúráját kiterjesztheti a mobil munkaerőre, növelve ezzel a kommunikáció és az információ feldolgozás hatékonyságát:

3.1 Leggyakrabban használt mobil szolgáltatások

Leggyakrabban a kommunikációs és információs eszközök használatosak, mint az e-mail, a forgalmi információk, útirány keresés, telefonkönyv, hírek, időjárás, város kalauz, naptár és feladatok feljegyzése, személyes feljegyzések, céginformációk, szótár, üdvözlet, vásárlási információk, sport, keresés, hírdetések, szórakozás de terjedőben vannak a vásárlói kereskedelmi eszközök is, mint a banki szolgáltatások, jegyvásárlás, fizetés, fogadás, befektetés, árurendelés.

4 Mobilbankolás bemutatása

4.1 A mobil bankolás előnyei

A legnagyobb előny, amit a mobil bankolás nyújt a bankoknak az, hogy drasztikusan lecsökkenti a szolgáltatások költségeit. Például egy átlagos bankpénztáros vagy telefonos tranzakció kb. 23-szor többe kerül egyenként, mint egy elektronikus tranzakció.

4.2 Leggyakrabban használt mobil banki ügyintézések és osztályozásuk

A bankok által leggyakrabban támogatott mobil banki ügyintézések:

- Számlaegyenleg lekérdezés
- Számla kivonat kérés
- Csekk állapot lekérdezés
- Csekk könyv kérés
- Átutalás számlák között
- Jóváírásra/terhelésre figyelmeztetés
- Minimum egyenleg figyelmeztetés
- Számla fizetési figyelmeztetés
- Számla fizetés
- Legutóbbi tranzakció történet lekérdezés
- Információkérés, mint a kamatláb/ devizaárfolyam.

Ezen szolgáltatások osztályozásának egyik módja a ‘Push/Pull' jelleg, ami egy szolgáltatási fázis kezdeményezőjétől függ. A ‘Push' az, mikor a bank információt küld ki megegyezés szerinti szabályok alapján, például a bank figyelmeztetést küld mikor a számlaegyenleg egy küszöbérték alá megy. A ‘Pull' az, mikor az ügyfél határozattal egy szolgáltatást vagy információt kér a banktól, így az utolsó öt tranzakció állapotának lekérdezése egy Pull alapú kérés.

A mobil bankolási szolgáltatások kategorizálásának másik módja a szolgáltatás jellege által, kétfajta szolgáltatást ad – tranzakció és lekérdezés alapú. Így egy banki bizonylat kérése lekérdezés alapú szolgáltatás és átutalás egyik számláról egy másikra tranzakció alapú szolgáltatás. A tranzakció alapú szolgáltatások szintén megkülönböztethetők a lekérdezés alapú szolgáltatásoktól abban az értelemben, hogy további biztonsági kéréssel rendelkeznek a csatornán keresztül a mobil telefontól a banki adatszerverekig.

A fenti osztályozások alapján elérkeztünk a következő rendszertanig.

	Push alapú	**Pull alapú**
Tranzakció alapú		• Tőke átutalás • Számla fizetés • Más pénzügyi szolgáltatások mint a részvénykereskedelem.
Lekérdezés alapú	• Tartozási/követelési figyelmeztetés. • Egyenleg minimum figyelmeztetés • Számla fizetési figyelmeztetés	• Számla egyenleg lekérdezés • Számla állapot lekérdezés • Csekk állapot lekérdezés • Csekk könyv kérések • Legutóbbi tranzakciók története.

4.3 Mobil bankolásra alkalmas főbb technológiák

A mobil bankoláshoz fejlesztett mobil alkalmazásokat a következő négy főbb csatornán lehet megvalósítani:

- IVR (Interactive Voice Response)
- SMS (Short Messaging Service)

- WAP (Wireless Access Protocol)
- Standalone Mobile Application Clients

4.3.1. IVR

IVR vagy Interactive Voice Response (interaktív hang válasz) szolgáltatás előre meghatározott számoknál működik, amelyet a bank hirdet az ügyfeleinek. Az ügyfél felhívja az IVR számot és általában egy tárolt elektronikus üzenet fogadja, amit különböző opciók menüje követ. Az ügyfelek választhatnak a megfelelő szám lenyomásával a billentyűzetükön, és akkor felolvasásra kerül a megfelelő információ.

Az IVR-en alapuló mobil bankolásnak néhány fő korláta, hogy csak lekérdezés alapú szolgáltatásokra lehet használni. Szintén költségesebb az IVR összehasonlítva más csatornákkal, minthogy magában foglal egy hangos hívást, amely általában sokkal drágább, mint egy SMS küldése vagy adatátvitel (mint a WAP-on vagy Standalone kliensek).

Az IVR elérés egyik módja egy PBX rendszer telepítése, ami fogadni tudja az IVR hívási terveket. A bankok az alacsony költségű módot keresik, ilyen az Asterix, amely egy nyílt forrású Linux PBX rendszer.

4.3.2. SMS – Short Messaging Service

Ennek a módja az, hogy az ügyfél információt kér egy SMS küldésével, ami szolgáltatási parancsot tartalmaz egy meghatározott számra. A bank egy válasz SMS-el válaszol, ami a speciális információt tartalmazza.

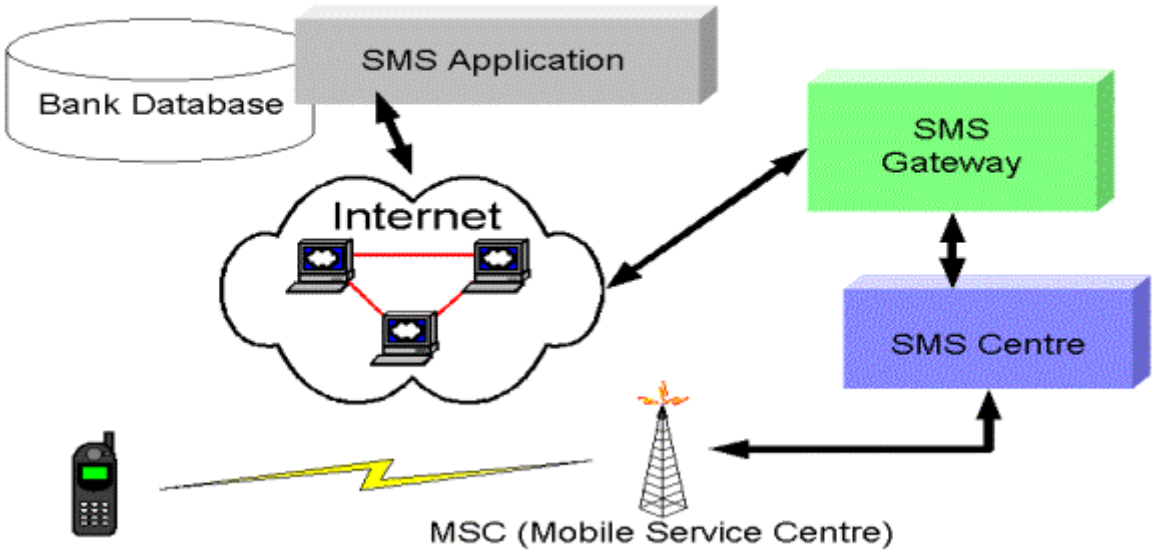
Például, a HDFC Bank ügyfelei Indiában megkaphatják a számla egyenlegük részleteit a ‘HDFCBAL' kulcsszó küldésével és ugyancsak SMS-ben megkapja az egyenleg információját. A szolgáltatások legtöbbje, amit a fő bankok SMS-el használnak, a lekérdezés alapú szolgáltatásokra korlátozódnak.

Habár van néhány eset, ahol még tranzakció alapú szolgáltatások is végezhetők SMS-el. Például a Bank of Punjab ügyfelei ‘ TRN(A/c No)(PIN No)(Amount)' SMS küldésével át is utalhatnak pénzösszegeket.

Az egyik fő oka, amiért a tranzakció alapú szolgáltatások nem végezhetők SMS-en az a biztonság miatti aggódás és mert az SMS nem teszi lehetővé, hogy az ügyfél felhasználói felület megfelelő legyen komplexebb szolgáltatások elérésére mint például a tranzakció.

Az SMS-en kifejlesztett mobil alkalmazások fő előnye az, hogy majdnem minden mobil telefonon, beleértve az alacsony élettartamúakat, olcsókat is, amelyek a legnépszerűbbek olyan országokban mint India és Kína az SMS elérhető.

Egy SMS alapú szolgáltatást egy SMS gateway fogadja, amely tovább kapcsolódik a Mobil szolgáltatást ellátó SMS központba. Van néhány fogadott IP alapú SMS gateway, ami a piacon elérhető és szintén van néhány nyílt forrású, mint a Kannel.



4. ábra: SMS hálózati felépítés

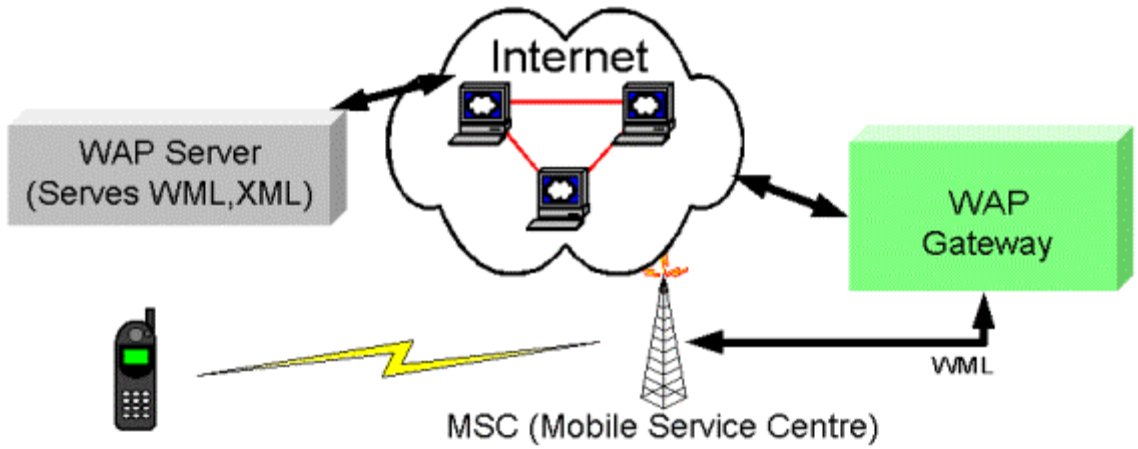
4.3.3. WAP – Wireless Access Protocol

A WAP hasonló koncepciót használ, mint az Internet bankolás. A bankok WAP oldalakat tartanak fenn, melyeket az ügyfelek egy WAP kompatibilis keresővel érnek el a mobil telefonukon. A WAP oldalak felhasználóbarát felületet nyújtanak és elég hatékonyan megvalósíthatók a biztonsági előírások.

A banki ügyfelek bárhol és bármikor elérhetnek egy megbízható biztonságú szolgáltatást, ami megenged minden lekérdezés és tranzakció alapú szolgáltatást és még további komplex tranzakciót, mint például az értékpapír kereskedelem telefonon keresztül.

Egy WAP alapú szolgáltatás egy befogadó WAP gateway-t igényel. A mobil alkalmazás használók a WAP gateway-en keresztül érik el a banki oldalakat, hogy végrehajtsák a tranzakciókat, mint ahogy az internet használók érik el a web portált a banki szolgáltatások hozzáféréséhez.

A következő ábra demonstrálja a WAP-on működő mobil alkalmazások keretrendszerét. Az aktuális űrlapok, amik egy mobil alkalmazásba mennek, egy WAP szerveren tárolódnak és igény szerint van kiszolgálás. A WAP Gateway kialakít egy hozzáférési pontot az internethez a mobil hálózatból.

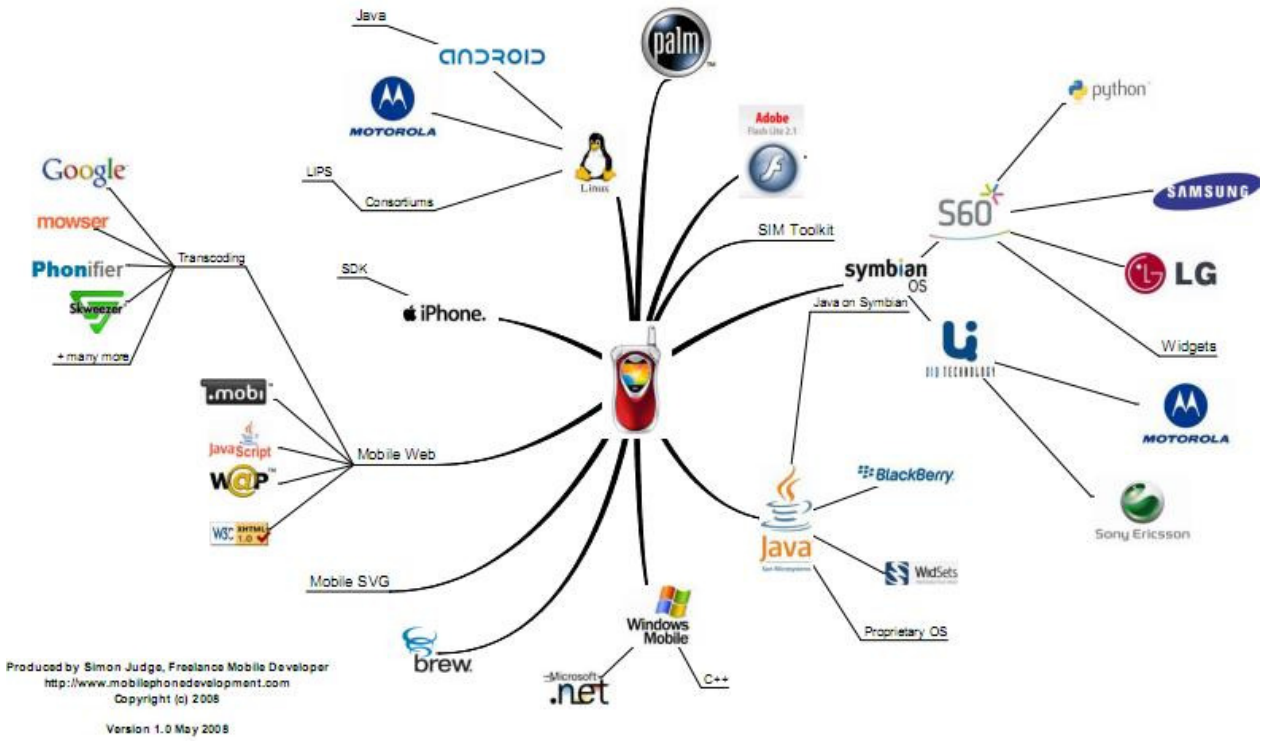


5. ábra: WAP hálózati felépítés mobil alkalmazásoknak

4.3.4. Standalone Mobile Application Clients

Standalone mobil alkalmazások az egyetlenek, melyeket a legígéretesebbnek tartanak, mivel a legalkalmasabbak komplex banki tranzakciók implementálására, mint az értékpapír kereskedelem. Könnyen testre szabhatók a felhasználó felület bonyolultsága szerint, amit támogat a mobil. Továbbá, a mobil alkalmazások képessé teszik a kommunikáció egy nagyon biztonságos és megbízható csatornájának az implementálását.

A mobil alkalmazások klienseinek egyik követelménye az, hogy a kliensen legyenek a használat előtt, ami továbbá azt igényli, hogy a mobil készülék támogasson a sok fejlesztői környezet közül egyet, mint a J2ME, Qualcomm's BREW vagy az Android. [4]



6. ábra: Mobil telefonokon található operációs rendszerek és azok programozási lehetőségei

A mobil alkalmazás kliensek fő hátránya, hogy az alkalmazásokat minden telefonra igény szerint kell kialakítani, amelyiken végül futhat. A J2ME összeköti az API-t azon mobil telefonok számára, melyeknek hasonló funkciói vannak, amiket ’profil’-nak hívnak. Azonban a mobil telefonok gyors szaporodása, melyek különböző funkciókat támogatnak, profilok óriási számát eredményezi, melyek további jelentősen növekedő fejlődési költségeket jelentenek. Ennek a problémának a mértéke felmérhető azon tény által, hogy a vállalatok által implementált mobil alkalmazás klienseknek több mint 50% költségre lehet szükségük a fejlesztési időre és a források igény szerinti kialakítására ahhoz, hogy az alkalmazásuk megfeleljen különböző mobil profilok igényeinek.

Az Android úgy tűnik, előnyben van most, mivel a Google elkészítette a szabadon használható fejlesztői eszközöket.

4.4 Kihívások a mobil bankolás megoldására

A kulcs feladatok egy kifinomult mobil bank alkalmazás fejlesztésben:

4.4.1. Együttműködési képesség

Hiány van a mobil bankolás közös technológia szabványokban. Számos protokolt használnak mobil bankolásra például HTML, WAP, SOAP, XML. Egy okos ötlet lenne az eladók számára egy olyan mobil alkalmazás kifejlesztése, ami több bankhoz tud kapcsolódni. Elvárható, hogy vagy az alkalmazás támogasson több protokolt, vagy a protokolok egy közös és széles körben fogadható beállítást használjanak az adatcserére.

Nagyszámú különböző mobil készülék van és nagy kihívás a bankoknak, hogy a készülékek bármelyik típusára mobil bankolási megoldást nyújtsanak. Néhány készülék a J2ME-t támogatja, míg mások WAP böngészőt vagy csak SMS-t.

Az igény az együttműködési képességre nagymértékben maguktól a bankoktól függnek, ahol az installált alkalmazások (Java alapú vagy más) nagyobb biztonságot nyújtanak, könnyebb használni és több komplex képesség fejlesztését engedi meg, hasonlóan az internet banki alkalmazásokhoz, míg az SMS az alapokat tudja nyújtani, de több komplex tranzakció végrehajtása bonyolult.

4.4.2. Biztonság

A következő vonatkozásokat kell megnevezni egy biztonságos infrastruktúrához a pénzügyi tranzakciók számára a vezeték nélküli hálózaton túl:

- A kézi készülékek fizikai része: ha a bank integrált hitelkártya alapú biztonságot ajánl, a készülék fizikai biztonsága még fontosabb.
- A készüléken futó kliens-közeli alkalmazások biztonsága: abban az esetben, ha a készüléket ellopják, a hackertől legalább kérhet egy ID/jelszót az alkalmazás eléréséhez.
- A készülék autentikációja a szolgáltatóval tranzakció kezdete előtt: ez biztosíthatja, hogy illetéktelen készülékek ne kapcsolódjanak pénzügyi tranzakció végrehajtásához.
- Banki ügyfelek felhasználói ID / Jelszó autentikációja.
- Adatok titkosítása az éterben való átvitelhez.
- Adatok titkosítása, amik a készülékben tárolódnak.

4.4.3. Megbízhatóság

A bankok másik kihívása az értékpapír rendszeres időközönként emelkedő árfolyamnál történő eladásához a mobil bankolás infrastruktúrája, hogy kezelni tudja az exponenciálisan növekvő ügyfél bázist. Az ügyfél a világ bármely részén mobil bankolhat (bármikor, bárhol) és ennélfogva a bankoknak biztosítani kell, hogy a rendszerek valós 24 x 7 módban fussanak. Ahogy az ügyfelek a mobil bankolást egyre hasznosabbnak találják, az elvárások is nőnek a megoldások felé. A bankok képtelenek összeegyeztetni a teljesítmény és megbízhatósági elvárásokat, amivel elveszíthetik az ügyfelek bizalmát.

4.4.4. Alkalmazás disztribúció

A bankok és ügyfeleik közötti összekapcsolhatóság természetének következtében nem lenne praktikus elvárni az ügyfelektől, hogy rendszeresen látogassák a bankot vagy kapcsolódjanak egy web oldalhoz a mobil bank alkalmazások rendszeres frissítéséhez. Elvárható, hogy a mobil alkalmazások maguk ellenőrizzék a frissítéseket és letöltse a szükséges darabokat (így Over The Air frissítéseknek hívják).

4.4.5. Személyesítés

A mobil alkalmazásoktól elvárható, hogy támogassák a személyessé tételt, úgy mint:

- Előnyben részesített nyelv
- Dátum / idő formátum
- Összeg formátum
- Hibás tranzakciók
- Kedvezményezetti lista
- Figyelmeztetések

5 Android bemutatása

Az Android-ot 2007 novemberében hozta nyilvánosságra az Open Handset Alliance (OHA) konzorcium, amelyet harmincnégy hardver, szoftver és telekommunikációs cég, köztük a Google, HTC, Motorola, Qualcomm, T-Mobile stb. alapított, a mobil készülékek nyílt szabványainak elősegítésére. Az Android az OHA által fejlesztett olyan mobil telefon platform, amely Linux operációs rendszerre épül. Az Android nem más, mint egy szoftver „verem" (software stack) mobil telefonokra, amely magában foglal egy operációs rendszert, egy middleware-t és a szükséges alkalmazásokat. Az Android SDK (Software Development Kit) ingyenesen letölthető (http://code.google.com/android/download.html), amely tartalmaz minden szükséges eszközt és API-t a Java nyelvű fejlesztéséhez.

Felmerülhet a kérdés, hogy milyen telefonok képesek az Android rendszer futtatására. Ellentétben az i-mode technológiával, open source projektről van szó, így lehetőség van a kernel (http://source.android.com/) és egyéb komponensek (http://code.google.com/p/android/downloads/list) szintén ingyenes letöltésére, így a készülék-gyártó cégek szabadon implementálhatják a saját Android platform-ot futtató telefonjukat.

A rendszer API-ja eszközt biztosít a készülék pontos helyének meghatározására, a SIM információk és funkciók elérésén át, a készülék saját böngészőjének programozásával bezárólag szinte minden, a készülék által nyújtott funkció eléréséhez.

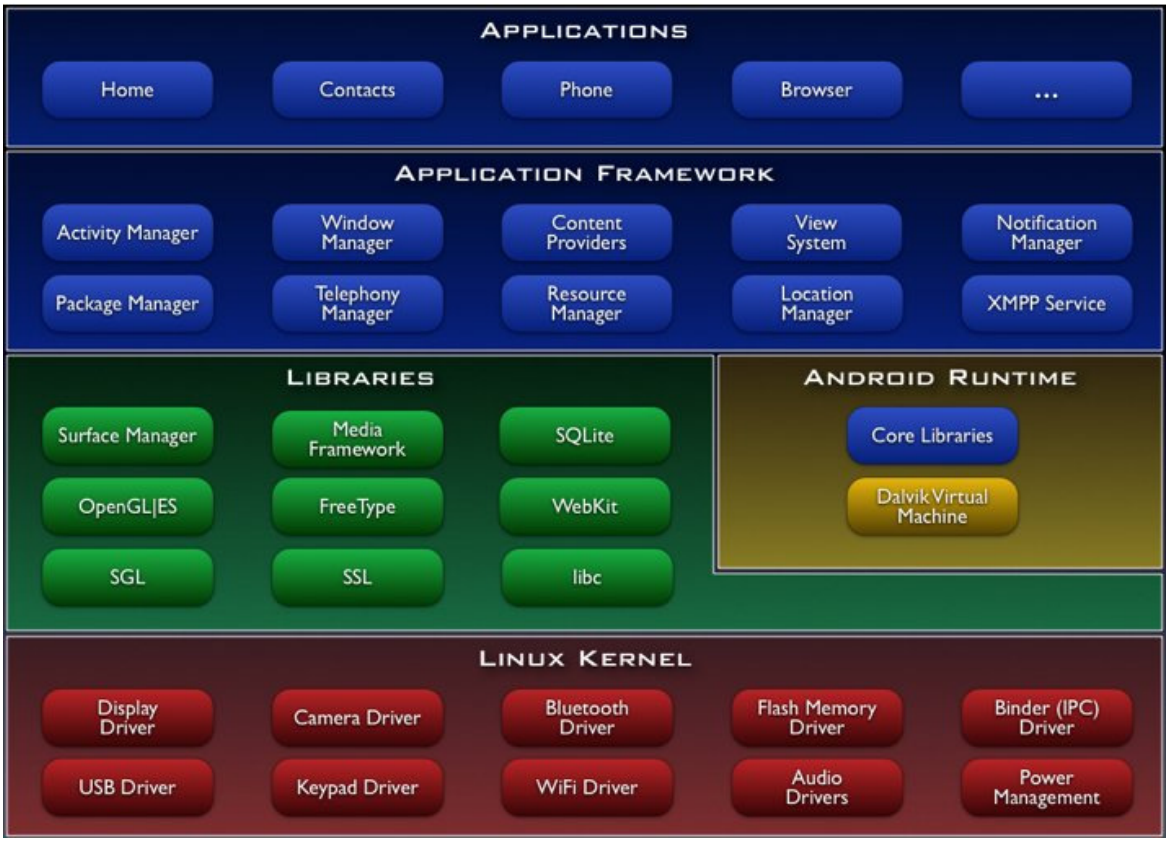
Ezeken felül az előző technológiák előnyeiből is számos jellemzi a rendszert. Például e technológia esetében is lehetőség van a titkosított adatátvitelre, vagy itt is kihasználható az érintő képernyő nyújtotta gyors funkció elérés. A böngésző alapú alkalmazások előnyeit is élvezhetjük az Android böngészőmoduljának beépítésével valamit az SMS küldés is lehetséges. Az alapértelmezett grafikus felhasználói felület komponensein kívül lehetőség van sajátos GUI elemek és úgynevezett témák implementálására is, amelyek az alkalmazások egységes kinézetéért felelnek. Továbbá eszközt biztosított a Google számos szolgáltatásának használatára is, mint például a Google Map szolgáltatás, amely segítségével térképek megjelenítésére, kezelésére és azokon történő rajzolásra is lehetőség van. Az Androidot úgy tervezték, hogy az általa futtatott alkalmazások lazán kapcsolhatóak legyenek. Ennek köszönhetően lehetőség van az egyes

programok között az adatok megosztására. Ezt, szolgáltatások formájában képesek megtenni. A fentiekből következően, az alkalmazások egymással kommunikáló modulokból épülhetnek fel.
Az Android API-jának fentebb kiemelt eszközeivel, olyan mobil banki kliens programok készíthetők, amelyek azon túl, hogy megfelelnek az alap biztonsági követelményeknek, gazdag felhasználói felülettel rendelkeznek, hatékonyak, több funkcionalitást nyújtanak, mint más technológiák. A következőkben röviden bemutatom az Android rendszer tulajdonságait és architektúráját.

Tulajdonságok:

- *Alkalmazás keretrendszer* (Application framework) a komponensek felcserélhetőségéért és újrahasznosíthatóságáért
- *Dalvik virtuális gép (DVM)* mobil eszközökre optimalizált virtuális gép
- *Integrált böngésző* a nyílt forrású WebKit engine-re alapozva
- *Optimalizált grafika* egyéni 2D grafikus könyvtárral, az OpenGL ES 1.0 specifikáción alapuló 3D grafikával (a hardveres gyorsítás opcionális)
- *SQLite* a strukturált adat tároláshoz
- *Média támogatás* általános audio és video formátumokra, valamint kép formátumokra (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- *GSM Telephony* (hardverfüggő)
- *Bluetooth, EDGE, 3G és WiFi* (hardverfüggő)
- *Kamera, GPS, iránytű és gyorsulásmérő* (hardverfüggő)
- *Gazdag fejlesztő környezet,* amely magába foglal egy készülék emulátort, eszközöket a hibakeresésre (debugging), memória és teljesítményhangolást, és plugin-t az Eclipse fejlesztő környezethez

Az Android operációs rendszer egy több rétegű, komplex rendszer, amely Linux kernelre épül. A következő diagram szemlélteti az Android operációs rendszer főbb komponenseit, amelyet a továbbiakban részletesebben is tárgyalni fogunk.



7. ábra: Az Android operációs rendszer architektúrája

5.1. Alkalmazások

Az Android számos előre beépített alkalmazást tartalmaz, mint például email kliens, SMS program, naptár, térképek, böngésző, kapcsolatok és még sok más. Minden alkalmazás Java programozási nyelven íródik.

5.2. Alkalmazás keretrendszer

A fejlesztők számára teljes mértékben elérhető az alapalkalmazások által használt keretrendszer API. Az alkalmazás architektúra úgy lett megtervezve, hogy egyszerűbbé teszi a komponensek újra hasznosíthatóságát. Minden alkalmazás közre tudja bocsátani a képességeit, és bármely más alkalmazás képes használni azokat (a keretrendszer által megkívánt korlátozások figyelembevételével). Ugyan ez a mechanizmus teszi lehetővé a felhasználónak, az egyes komponensek lecserélését.

Ennek alapján minden alkalmazás szolgáltatások és rendszerek halmaza, amely magában foglalja:

- Gazdag és kiterjeszthető nézetek (`View`) halmazát, amit az alkalmazások építése során lehet használni, például listák, táblák, szöveg mezők, gombok és még egy beágyazható web böngészőt is.
- Tartalomszolgáltatókat (`ContentProviders`), amelyek segítségével lehetőség nyílik más alkalmazásoktól származó adatok elérésére (például a kapcsolatok adatai), vagy saját adataik megosztására
- Erőforrás menedzsert, amely elérést biztosít a nem kód alapú erőforrásokhoz, például képek, fájlok
- Értesítési menedzsert, amely biztosítja minden alkalmazás számára, az egyedi értesítők megjelenítését az állapotsorban.
- Tevékenységi menedzsert, amely irányítja az alkalmazások életciklusait

5.3. Könyvtárak

Az Android számos C/C++ könyvtárt tartalmaz, amelyeket az Android rendszer különböző komponensei használnak. Ezen könyvtárak funkciói az alkalmazás keretrendszeren keresztül érhetők el a fejlesztők számára. Néhány ilyen könyvtár:

- *C rendszer könyvtár* – a standard C rendszer könyvtár (libc) BSD-származékú implementációja, beágyazott Linux alapú rendszerekre hangolva

- *Média Könyvtárak* – a PacketVideo OpenCORE-ján alapulnak; számos közkedvelt kép, audio és video formátumok lejátszását és rögzítését támogatják
- *Felület menedzser* – vezérli a kijelző alrendszer elérését, és lehetővé teszi több alkalmazás 2D-s és 3D-s grafikus layer-ének egyesítését.
- *LibWebCore* – egy modern web böngésző motor, amely mind az Android böngészőt, mind egy beágyazható web nézetet menedzselhet
- *SGL* – az alap 2D grafikus motor
- *3D könyvtárak* – az OpenGL ES 1.0 API-ra épülő implementáció; a könyvtárak vagy a hardver 3D gyorsítót használják (ha elérhető) vagy a beépített, optimalizált 3D szoftver raszterizálót
- *FreeType* – bitmap és vector font betű renderelés is biztosított
- *SQLite* – hatékony és könnyűsúlyú relációs adatbázis motor, amely minden alkalmazás számára elérhető

5.4. Android Runtime

Az Androidnak ez a komponense foglalja magába az alap könyvtárak azon halmazát, amelyek a legtöbb funkcionalitást biztosítják a Java programozási nyelv alap könyvtárai számára. Minden Android alkalmazás a saját folyamatában fut, az ő saját Dalvik virtuális gép példányával. A Dalvik virtuális gépet Bornstein nevezte el egy halászfalu, Dalvík in Eyjafjörður (Izland) után, ahol néhány őse élt. A Dalvikot azért írták, hogy az eszköz képes legyen több VM (Virtual Machine) hatékony futtatására. A Dalvik VM sajátos kiterjesztésű futtatható fájlokat (.dex) használ, amelyeket minimális memória használatra optimalizáltak. A regiszter alapú VM a beépített úgynevezett „dx” eszközzel .dex fájlokká átkonvertált, Java fordítóval készített osztályokat képes futtatni. A Dalvik VM a Linux kernelre támaszkodva ad lehetőséget a szálkezelésre és az alacsony szintű memóriamenedzselésre.

5.5. Linux kernel

Az Android a 2.6-os Linux-ra épül, amely elérhetővé teszi az olyan alap rendszerszolgáltatásokat, mint a biztonság, memóriamenedzselés, folyamatmenedzselés, hálózati stack, és a meghajtó

kezelés. A kernel absztrakciós rétegként is funkcionál (Hardware Abstraction Layer) a hardver és a szoftver stack között.

5.6. Android alkalmazások fejlesztése

Az Android alkalmazásaink fejlesztésére használhatjuk a már jól megszokott, jó minőségű Java fejlesztő környezeteinket. Azonban az Eclipse 3.2, vagy későbbi változatához elérhetővé tettek egy plugin-t (ADT plugin http://code.google.com/intl/hu-HU/android/adt_download.html), amely segítségével az Android alkalmazásainkat az Eclipse fejlesztő környezet hibakeresési és fejlesztési funkciói nyújtotta előnyökkel készíthetjük el. Az Android „core" könyvtárak biztosítják azokat a funkcionalitásokat, amelyek a „gazdag" mobil alkalmazások fejlesztéséhez szükségesek. Az Android fejlesztő eszközök (tools) pedig lehetővé teszik alkalmazásaink futtatását, hibakeresését és tesztelését. Az Android könyvtárakon kívül használhatóak az alap Java csomagok is.

5.7. Az Android alkalmazások sajátosságai

Mint minden mobil szovtertechnológiával fejlesztett alkalmazásnak, az Androiddal készített alkalmazásoknak is megvannak a maguk sajátosságai, viszont számos ponton hasonlíthatnak elődjeikre. Ezek a sajátosságok lényegében az alkalmazások szerkezeti felépítésében, és életciklusaiban rejlenek. Egy Android alkalmazás hibamentes működéséhez elengedhetelen a jól megválasztott komponensek használata. Ebben a fejezetben részletesebben kitérek az Android alkalmazások azon tulajdonságaira, amelyek nagyban befolyásolják a megvalósítandó program futásídejű működését.

5.7.1. Egy Android alkalmazás felépítése

Az Android alkalmazások négyféle építőelemből állnak:

- Activity
- Intent vevő
- Service
- Tartalomszolgáltató

Nem szükséges ezek mindegyikét felhasználni egy működő alkalmazás elkészítéséhez, viszont minden Android alkalmazás ezek valamilyen kombinációjaként épül fel. Ha sikerül meghatározni, hogy milyen komponensek használata szükségesek az adott alkalmazáshoz, fel kell sorolni őket az AndroidManifest.xml fájlban. Ez egy olyan XML fájl, amiben deklarálni kell az adott alkalmazás által használt komponenseket, valamint azok lehetőségeit és szükségleteit.

5.7.1.1. Activity

Ez a leggyakrabban használt építőelem. Egy `Activity` általában az alkalmazás egyetlen képernyője. Minden `Activity` egy olyan egyszerű osztályként van implementálva, amely az `Activity` alaposztály leszármazottja. Ezek az osztályok az úgynevezett `View`-okból álló felhasználói felület megjelenítéséért és a felhasználói események kezeléséért felelnek. A legtöbb alkalmazás több képernyőből áll. Például egy üzenetküldő alkalmazásnak lehet egy képernyője, amin ki lehet választani, hogy kinek szóljon az üzenet, valamint egy másik képernyője, amin megszerkeszthető. Mindkét képernyőt egy-egy `Activity`-ként lehet implementálni. Másik képernyőre való navigálás egy új `Activity` elindítását eredményezi. Előfordulhat, hogy adott `Activity` -nek vissza kell térnie egy értékkel a hívó `Activity`-ben. Például egy fénykép kiválasztó `Activity`-nek vissza kell térnie a kiválasztott fényképpel az őt hívónak. Egy új képernyő megnyitásakor az előző képernyő futása felfüggesztődig (pause), és a képernyő bekerül az előzmény verembe (history stack). A képernyőknek lehetőségük van kérni, hogy eltávolítódjanak az előzmény veremből, ha valamilyen oknál fogva nem szabad ott maradniuk. Az Android minden a home képernyőről indított alkalmazás számára fenntart egy előzmény vermet.

5.7.1.2. Intent és Intent szűrők

Android egy speciális úgynevezett `Intent` osztályt használ a képernyők közötti mozgáshoz. Egy `Intent` leírja, hogy egy alkalmazás mit akar tenni. Az `Intent` adatstruktúrájának két legfontosabb része a tevékenység (action) és az adat, amire a tevékenység hatással van. Tipikus értékek a tevékenységre a `MAIN` (az `Activity` belépő, indító pontja), `VIEW`, `PICK`, `EDIT`, stb.. Az adat URI formájában van kifejezve. Például egy személy kapcsolati információinak

megtekintésére egy Intent-et kell készíteni VIEW tevékenységgel, és az adott személyt reprezentáló URI adattal.

Egy másik idetartozó osztály az IntentFilter osztály. Míg egy Intent valójában a tevékenység igénye, addig az IntentFilter egy leírás, hogy egy Activity milyen Intent-ek (vagy Intent vevők (IntentReceiver)) kezelésére képes. Egy személy kapcsolati információi megjelenítésére képes Activity közzétehet egy IntentFilter-t, amely leírja, hogy az Activity tudja, hogyan kezelje a VIEW tevékenységet, amikor azt egy személy adatainak megjelenítésére alkalmazzák. Az Activity-k az IntentFilter-üket az AndroidManifest.xml fájlukban publikálják.

A képernyőről-képernyőre történő navigálás az Intent-ek feloldásával valósul meg. Az előrehaladáskor az aktuális Activity meghívja startActivity(myIntent) metódusát. A rendszer ezután megvizsgálja az összes telepített alkalmazás IntentFilter-ét, és kiválasztja azt az Activity-t, amelynek az IntentFilter-eiben leírtak legjobban egyeznek a myIntent-el. Ezután az új Activity értesítést kap az Intent-ben igényelt tevékenységről, aminek hatására elindítódik. Az Intent-ek feloldása futásidőben történik, ami két kulcsfontosságú előnyel is jár. Az Activity-k egyszerűen, Intent formába öntött kérésekkel képesek újra felhasználni más komponensek funkcionalitásait. Az Activity-k akármikor lecserélhetőek egy ugyanolyan IntentFilter használatával.

5.7.1.3. Intent vevő

Az IntentReceiver eszközt biztosít, hogy külső események hatására futtatódjon egy Activity, például telefon csengéskor, vagy amikor elérhető az adathálózat, vagy mondjuk éjfélkor. Az Intent vevőknek nincs grafikus felhasználói felületük, azonban használhatják az értesítés menedzsert (NotificationManager), hogy értesítést küldjenek a felhasználónak, az érdekes események bekövetkezéséről. Az Intent vevőket szintén az AndroidManifest.xml fájlba kell regisztrálni, de a Context.registerReceiver() metódussal lehetőség van programból történő regisztrálásra is. Az indítani kívánt alkalmazásnak nem kell futnia, hogy meghívódjon az IntentReceiver-je; a rendszer indítja el az alkalmazást, ha az Intent vevő megkapja a kiváltó jelet (triggering). A Context.broadcastIntent() metódussal üzenetszórás formájában az alkalmazások is küldhetnek ilyen kiváltó jelet.

5.7.1.4. Service

A `Service` egy kód, ami hosszú élettartamú és GUI nélkül fut. Jó példa erre, amikor egy média lejátszó zeneszámokat játszik. Egy médialejátszó alkalmazásban valószínűleg egy vagy több `Activity` is van, hogy a felhasználó kitudja választani, és futtatni tudja a kívánt médiát. Azonban a zene lejátszását nem szabad egy `Activity`-re bízni, hiszen a felhasználó arra számít, hogy a zene lejátszása akkor is folytatódik, ha ő egy másik képernyőre navigál. Ilyen esetben indíthatunk egy szolgáltatást (`Service`) a háttérben, a `Context.startService()` metódussal, hogy a zene lejátszása ne szakadjon meg. A rendszer ezután addig futtatja a zenét, amíg annak vége nem lesz, vagy le nem állítjuk azt. Lehetőség van továbbá egy már futó `Service`-hez kapcsolódni (vagy elindítani, ha az még nem fut), a `Context.bindService()` függvényhívással. Egy szolgáltatáshoz való kapcsolódás után, annak interfészén keresztül lehet vele kommunikálni. Az előbbi példaalkalmazásnál maradva, ezen keresztül tudunk a következő számra váltani, leállítani stb.

5.7.1.5. Tartalomszolgáltató

Az alkalmazások fájlban, SQLite adatbázisban és más értelmes tárolási mechanizmussal tárolhatnak adatot. Egy tartalomszolgáltató (`ContentProvider`) akkor hasznos, ha egy alkalmazás adatait meg szeretnénk osztani más alkalmazásokkal. Alapból minden alkalmazás csak a saját adataihoz fér hozzá, legyen az akár fájlban, akár adatbázisban. A tartalomszolgáltató egy osztály, amellyel más alkalmazások képesek a tartalomszolgáltató által kezelt adattípusnak megfelelő adatot kinyerni és tárolni.

5.7.2. Android felhasználói felületek

Az Androidban a felhasználói felületeket két módon építhetjük fel, XML –ből vagy Java-kód írásával. Az XML-ben definiált GUI struktúra preferáltabb, mivel a Model-Viewer-Control elvből ismert, hogy az UI mindig el kell különüljön a program-logikától. Egy UI definiálása XML-ben nagyon hasonlít egy HTML dokumentum készítéséhez, például:

```
<html>
   <head>
      <title>Page Title</title>
   </head>
   <body>
       The content of the body element.
   </body>
</html>
```

Ugyanez az Android XML-elrendezésében. Minden jól struktúrált és kifejezhető fa-szerkezettel:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
   <TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Hello World"/>
</LinearLayout>
```

5.7.2.1. Képernyő elemek hierarchiája

1. **Activity-k**

Egy Android alkalmazás alap funkció egysége az `Activity` – az android.app.Activity osztály egy objektuma. Egy `Activity` sok dolgot meg tud csinálni, de saját magában nem tud megjeleníteni a képernyőn. Ahhoz hogy egy `Activity` felületet jelenítsen meg, `view` és `viewgroup`-okkal kell dolgoznia – a felhasználói felület kifejezésének alap egységei az Android platformon.

2. **Az Activity-k életciklusa**

Az `Activity` fontos része egy alkalmazás életciklusának, ezért részletesebben is kitérek az ismertetésére. A rendszerben az `Activity` -k `Activity` veremként vannak menedzselve. Egy új `Activity` indulásakor a verem tetejére kerül és ő lesz az aktuális futó `Activity`. Az előző `Activity` mindig alul marad a veremben, és legközelebb már csak akkor kerül még egyszer előtérbe, amikor az új `Activity` befejeződött.

Egy `Activity` -nek négy státusza lehet:

- Ha egy `Activity` a képernyőn előtérben van (a verem tetején), akkor az `Activity` *aktív*, vagy *futó*.
- Ha egy `Activity` elvesztette a fókuszt, de még látszik (ez az, amikor egy nem teljes-képernyős, vagy transzparens `Activity` nyílik az eredeti `Activity` felé), akkor *pause* státuszba kerül. Egy *pause* státuszban lévő (felfüggesztett) `Activity` teljesen élő (megőrzi aktuális állapotát, és kapcsolatban marad az ablak menedzserrel), de kritikus memória hiány esetén terminálhatja a rendszer

- Ha egy `Activity`-t teljesen elfed egy másik Activity, akkor az elfedett státusza *stopped* lesz. Ez esetben is megőrződik az `Activity` állapota, de mivel már nem látszódik, így a rendszer bármikor terminálhatja, ha memória hiány lép fel.

- Ha egy `Activity` *pause* vagy *stopped* státuszú, a rendszer vagy megkéri, hogy fejezze be működését, vagy egyszerűen terminálja a processzt, amiben fut. Ha a felhasználó ismét erre a képernyőre navigál, akkor az `Activity` teljes újraindítása és az előző állapot helyreállítása szükséges.

Az `Activity`-k élettartamának három fő szakasza különböztethető meg:

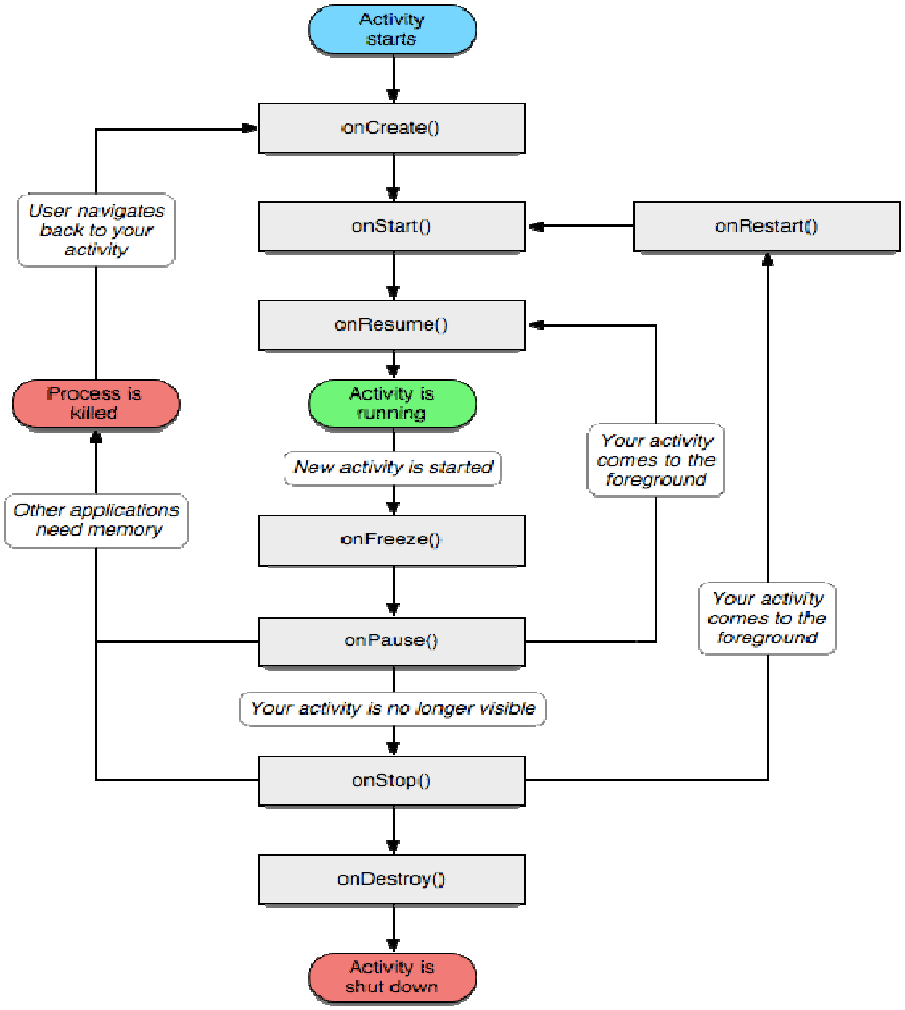
- Egy `Activity` **teljes élettartama** az `onCreate(Bundle)` függvény első hívásától az egyetlen és utolsó `onDestroy()` függvényhívásáig tart. Egy `Activity` a globális állapotának beállítását az `onCreate()` függvényben végzi, míg a lefoglalt erőforrásokat az `onDestroy()` metódusban szabadítja fel. Például egy, a hálózatról adatokat letöltő, háttérben futó szálat az `onCreate()` metódusban kell kreálni és az `onDestroy()` metódusban kell leállítani.

- Egy `Activity` **látható élettartama** az `onStart()` függvényhívástól az `onStop()` függvényhívásig tart. Ez idő alatt a felhasználó láthatja az `Activity`-t a képernyőn, bár

interakcióba még nem tud vele lépni. A két függvényhívás között lehet azokkal az erőforrásokkal törődni, amiket az `Activity`-nek meg kell jelenítenie a képernyőn. Például az `onStart()` metódusban készíthető egy `IntentReceiver`, amivel figyelhetőek bizonyos változások, aminek a hatására a felhasználói felületen módosítások végezhetők. Később, az `onStop()` függvényben a módosított GUI visszaállítható az eredeti állapotába. Mindkét függvény többször meghívható annak függvényében, hogy az `Activity` látható, vagy nem látható.

- Egy `Activity` **előtérben eltöltött élettartama** az `onResume()` és az `onPause()` függvényhívások között zajlik. Ez idő alatt az `Activity` minden más `Activity` előtt helyezkedik el a képernyőn, és képes interakciót végezni a felhasználóval. Egy `Activity` gyakran válthat a *resumed* és *paused* státuszai között - például, ha az eszköz készenléti állapotba kerül, vagy amikor egy `Activity` eredménye, vagy egy új Intent kézbesítődik – éppen ezért ezekben a metódusokban lényegesen kevesebb kódot szabad elhelyezni.

A következő ábra szemlélteti az `Activity` -k fontosabb státuszait és a köztük lévő átmeneteket. A téglalapok egy-egy callback metódust reprezentálnak, amelyeket implementálva meghatározható, hogy egy `Activity` az egyes státuszai közti átmenetekben milyen műveleteket hajtson végre. A színezett oválisok az `Activity` fő státuszait jelentik.



8. ábra: Az Activity-k életciklusa

Egy `Activity` teljes életciklusát a következő metódusok definiálják. Azért, hogy az egyes státuszok közötti váltáskor a megfelelő műveletek végrehajtódjanak, felül kell definiálni őket. Minden `Activity`-nek implementálnia kell az `onCreate()` metódusát, amelyben a kezdeti beállítások végzendők. Sokban felül kell definiálni az `onPause()` metódust, amelyben az adatok módosításainak véglegesítését (commit) szokás elvégezni, vagy előkészíthető az Activity a felhasználóval való interakció befejezésére. Általában sok `Activity`-nek meg kell valósítania az `onFreeze(Bundle outState)` metódusát, amelyben az `Activity` bizonyos állapot információit lehet elmenteni. Ez lehet például egy szám, amely egy lista aktuális elemének sorszáma, vagy egy kurzor pozíció egy szövegmezőben, viszont nem tartalmazhat felhasználói adatokat. A megfelelő állapot-helyreállítás az `onCreate(Bundle icicle)` metódussal végezhető. A többi metódust szintén szükség szerint lehet implementálni.

3. View-ok

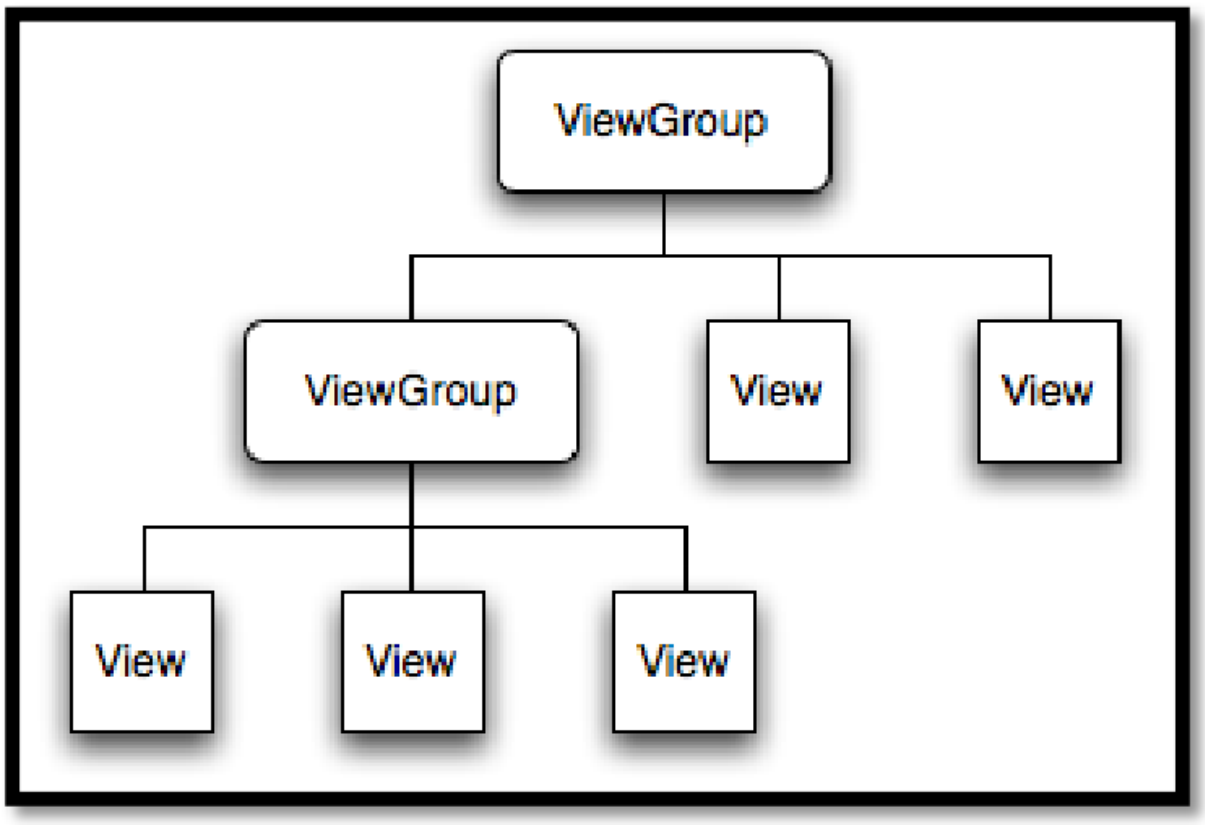
Egy `view` egy objektum, az android.view.View alaposztály kiterjesztése. Ez egy adat struktúra, ahol a tulajdonságok tárolják az elrendezést és a képernyő egy speciális négyzet területét tartalmazza. Egy `View` objektum kezeli a méréseket, az elrendezését, rajzolást, fókusz változást, görgetést és kulcsot a képernyő területhez, amit megjelenít. A `View` osztály alaposztálya minden eszköznek – egy halmaza a teljesen implementált alosztályoknak, ami kirajzolja az interaktív képernyő elemeket. Az eszközök kezelik a saját méréseiket és rajzolásukat, így fel tudjuk használni a saját UI sokkal gyorsabb építéséhez. Néhány `view`: TextView, EditText, Button, RadioButton, Checkbox, ScrollView, ….

4. Viewgroup-ok

Egy `viewgroup` egy objektum, az android.view.Viewgroup osztály kiterjesztése. Ahogy a neve is mutatja, egy `viewgroup` egy speciális típusa a `view` objektumnak, amely funkciója az, hogy tartalmazzon és irányítson egy alárendelt halmazát a `view`–knak és más `viewgroup`–oknak, a `Viewgroup`–okat hozzá lehet adni a felületi struktúrához és komplex képernyő elemeket felépíteni, ami megcímezhető, mint egy egyszerű entitás. A `Viewgroup` osztály alaposztálya az elrendezéseknek – egy halmaza a teljesen implementált alosztályoknak, ami a képernyő elrendezés közös típusát adja. Az elrendezések módot adnak a `view`–ok egy halmazának szerkezeti felépítéséhez.

5. A fa-struktúrált UI

Az Android platformon, egy `Activity` UI-ját `view` és `viewgroup`–ok egy fa szerkezeteként definiáljuk, mint ahogy a diagram mutatja. A fa lehet egyszerű vagy összetett, amire szükség van, és felépíthető Android elődefiniált eszközeivel és elrendezéseivel, vagy testreszabható `view` típusokkal, amit saját magunk készítettünk.



9. ábra: Android UI – Fa struktúra

A fához csatolásnál az `Activity` meghívja a `setContentView()` metódusát és referenciát ad át a gyökér csomópont objektumnak. Az Android rendszernek referenciája van a gyökér csomópont objektumhoz, közvetlenül tud dolgozni a csomóponttal, hogy érvénytelenítsen, mérjen és kirajzolja a fát. Mikor az `Activity` aktívvá válik és figyelmet kap, a rendszer értesíti az `activity`-t és kéri a gyökér csomópontot, hogy mérje és rajzolja meg a fát. A gyökér csomópont ekkor azt kéri, hogy a gyerek csomópontontjai rajzolják ki magukat – minden `viewgroup` csomópont a fában közvetlenül felelős a gyerekeik kirajzolásáért. Minden `viewgroup` felelős az elérhető terület méréséért, a gyerekek elrendezéséért, és a `draw()` függvény meghívásáért minden gyereken, hogy megjelenjenek. A gyerekek kérhetnek méretet és elhelyezkedést a szülőben, de a szülő objektumé a végső döntés, hogy hol milyen nagy lehet minden gyerek.

5.8. Az AndroidManifest.xml

Az `AndroidManifest.xml` minden Android alkalmazáshoz szükséges. Az alkalmazás gyökér könyvtárában helyezkedik el, és globális értékeket ír le a csomaghoz az alkalmazás komponenseit (`activity`, `service`, stb) tartalmazva amit a csomag kitesz. Továbbá, minden komponens számára leírja az implementációs osztályokat, ilyen adatokat tud kezelni, és hogy tudja elindítani. Fontos megemlíteni, hogy ezek úgynevezett `IntentFilter`-k. Az alkalmazás `Activity`, `Content Providers`, `Services`, és `Intent Receivers` deklarálása mellett specifikálhatóak jogosultságok az `AndroidManifest.xml`-ben.

Egy nagyon egyszerű `AndroidManifest.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="org.anddev.android.hello_android">
   <application android:icon="@drawable/icon">
     <activity android:name=".Hello_Android"
     android:label="@string/app_name">
       <intent-filter>
         <action android:name="android.intent.action.MAIN" />
         <category android:name="android.intent.category.LAUNCHER"/>
       </intent-filter>
     </activity>
   </application>
</manifest>
```

- Minden `AndroidManifest.xml` tartalmazza a névtér deklarációját az első elemében.

```
xmlns:android=http://schemas.android.com/apk/res/android
```

- Ez szabványos Android attribútumok választékát teszi elérhetővé a fájlban, amit arra használunk, hogy az elemek számára a legtöbb adatot szolgáltassa.

- Minden manifest egyetlen `<application>` tag-et foglal magában, ami saját maga számos taget tartalmaz, leírva az `Activity`, `IntentReceiver`, stb… ami elérhető ebben az alkalmazásban.

- Ha egy Activityt akarsz csinálni, ami közvetlenül a felhasználón keresztül indítható, akkor támogatni kell a `MAIN` tevékenységet és `LAUNCHER` kategóriát.

Egy részletes lista következik egy `AndroidManifest` fájl struktúrájáról, ami minden elérhető tag-et egy példán keresztül mutat be:

<manifest>

Ez minden `AndroidManifest.xml` gyökér csomópontja. Tartalmazza a csomag-attribútumot, ami bármely csomagra rámutat. Az `Activity`-k elérhetők ettől az értéktől relatívan.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="org.anddev.android.smstretcher">
```

<uses-permission>

Egy biztonsági engedélyt ír le, amit a csomagon engedélyezni kell a megfelelő működéshez (pl. SMS küldés vagy kapcsolati lista használata). A jogosultságokat a felhasználó engedélyezi az alkalmazás telepítése alatt. Mennyiség: 0+

```
<uses-permission android:name=" android.permission.RECEIVE_SMS"/>
```

<permission>

Egy biztonsági engedélyt deklarál, ami arra használható, hogy korlátozza, melyik alkalmazások érhetik el a csomag komponenseit vagy tulajdonságait. Mennyiség: 0+

<instrumentation>

Egy `instrumentation` komponens kódját deklarálja, ami arra szolgál, hogy tesztelje a csomag funkcionalitását. Mennyiség: 0+

<application>

Gyökér elem, ami tartalmazza az alkalmazás-szintű komponensek deklarációját, amik a csomagban vannak. Ezek az elemek szintén tartalmazhatnak globális és/vagy alapértelmezett attribútumokat az alkalmazás számára, mint egy címke, ikon, téma, kívánt engedély, stb. Mennyiség: 0 vagy 1.

```
<application android:icon="@drawable/icon">
```

A következő gyerekek mindegyik 0+ -át elhelyezheted:

- ***<activity>***

Egy alkalmazás számára az elsődleges dolog egy `Activity`, hogy a felhasználóval egymásra hasson. A kezdeti képernyő, amit a felhasználó lát egy alkalmazás indításakor egy `activity`, és a többi képernyőhöz, amit implementálni fogunk további `activity` tag-eket kell deklarálni, mint elkülönített `activity`-k.

```
<activity android:name=".Welcome" android:label="@string/app_name">
```

Megjegyzés: *Minden `Activity`-hez tartozik egy `<activity>` tag a manifest-ben, hogy csak a saját csomagjában legyen használható. Ha egy `Activity`-hez nem található tag a manifestben, akkor nem fordítható le.*

Feltételesen, az `activity` tag magában kell foglaljon egy 1+ számosságú `<intent-filter>` elemeket, ami leírja a tevékenységeket az `activity` támogatására.

- ***<intent-filter>***

Deklarálja, hogy milyen fajta `Intent` komponenseket támogat. Továbbá a különböző típusú értékeket, ami ez alatt az elem alatt specifikálható, attribútumként megadható egy egyedi címke, ikon és más információ az akció leírásához.

- ***<action>***

Egy akció típus, amit a komponens támogat. Például:

```
<action android:name="android.intent.action.MAIN" />
```

- ***<category>***

Egy kategória-típus, amit a komponens támogat. például:

```
<category android:name="android.intent.category.LAUNCHER" />
```

- ***<data>***

Egy MIME típus, URI elrendezés, URI engedély, vagy URI út, amit a komponens támogat. 1+ számosságú metaadat társítható az activityhez:

- **<meta-data>**

Egy új darab meta adat hozzáadása az `activity`-hez, amit a kliens kinyerhet a `ComponentInfo.metaData`-n keresztül.

- **<receiver>**

Egy `IntentReceiver` megengedi egy alkalmazásnak, hogy beszéljen a változásokról vagy akciókról, amik történnek, még ha jelenleg nem is fut.Mint az `activity tag`-nél,tartalmazhat 1+ számosságú `<intent-filter>` elemeket, amit a `receiver` támogat vagy `<meta-data>` értékeket, úgyanúgy mint az `<activity>`-nél.

```
<receiver android:name=".SMSReceiver">
```

- **<service>**

Egy `Service` egy komponens, ami korlátlan ideig futhat a háttérben. Mint az `activity tag`-nél, egy vagy több `<intent-filter>` elemeket tartalmazhat, amit a `service` támogat vagy `<meta-data>` értékeket; lásd az `activity <intent-filter>` és `<meta-data>` leírást.

- **<provider>**

Egy `ContentProvider` egy komponens, ami kezeli a perzisztens adatokat és publikálja más alkalmazások számára. Egy vagy több `<meta-data>` érték csatolható.

Természetesen minden `tag`-nek kell lenni egy lezáró `tag`-jének.

5.9. Források és a varázslatos R.java

Egy projekt forrásai és az R.java nagyon közel állnak egymáshoz.

5.9.1. Források

A források külső fájlok (nem-kód fájlok), amit a kód használ és belefordítódik az alkalmazásba felépítéskor. Az Android számos különböző fajta forrás fájlt támogat, beleértve az XML, PNG, és JPEG fájlokat. Az XML fájlok formátuma különböző.

A források kívül esnek a forráskódtól, és az XML fájlok egy bináris, gyorsan tölthető formátumra fordulnak. A String-eket egy sokkal hatékonyabban tárolható formátumra tömöríti.

5.9.1.1. A források listája

Forrás-típusok és ahol elhelyezkednek:

- layout-fájlok `"/res/layout/"`
- képek `"/res/drawable/"`
- animációk `"/res/anim/"`
- stílusok, string-ek és tömbök `"/res/values/"`

Nevek amelyeknek nem kell, hogy pontosan ilyenek legyenek:

  - arrays.xml tömbök definiálásához
  - colors.xml színek definiálásához
    - `#RGB, #ARGB, #RRGGBB, #AARRGGBB`
  - dimens.xml méretek definiálásához
  - strings.xml stringek definiálásához
  - styles.xml stílus objektumok definiálásához
- Nyers fájlok mint mp3-ak vagy videók `"/res/raw/"`

5.9.1.2. Források használata a kódban

Források használata a kódban csak a teljes forrás ID ismeretében lehet. A forrás hivatkozásának szintaxisa:

```
R.resource_type.resource_name
```

vagy

```
android.R.resource_type.resource_name
```

Ahol a `resource_type` az R alosztály, ami `resource.resource_name` specifikus típusát tartja, XML fájlokban definiált források név attribútuma, vagy más fájltípusokban definiált források fájl neve (kiterjesztés nélkül). A források mindegyik típusa hozzáadódik egy specifikus R alosztályhoz a forrás típusától függően. A saját alkalmazás által szerkesztett forrásra csomag név nélkül lehet hivatkozni (egyszerűen mint `R.resource_type.resource_name`). Az Android számos szabvány forrást tartalmaz, úgy mint a képernyő stílus és gomb háttér. Ezek azonosításához a kódban, minősíteni kell őket az `android.`-dal, például:

```
android.R.drawable.button_background.
```

5.9.1.3. Referencia források

Egy érték, ami egy attribútumban van (vagy forrásban) szintén lehet egy forrás referenciája. Ezt gyakran a layout fájlokban használják a string-ek ellátására (tehát helyhez kötöttek lehetnek) és képek (amik másik fájlban léteznek), habár egy referencia bármely forrás típus lehet beleértve a színeket és integereket. Például, ha szín forrásaink vannak, írhatunk egy layout fájlt, ami beállítja a szöveg szín formátumát, hogy olyan érték legyen, amit az egyik forrás tartalmaz:

```
<EditText
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:textColor="@color/opaque_red"
android:text="Hello, World!" />
```

Meg kell jegyezni, hogy a '@' prefixum használatával bevezethető egy forrás referencia – a szöveget követve, ami egy forrás neve `@[package:]type/name` formában. Ebben az esetben

nem volt szükség specifikálni a csomagot, mert hivatkozunk egy forrásra a saját csomagunkban. Egy rendszer forrásra hivatkozáshoz írni kellene:

```
<EditText
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:textColor="@android:color/opaque_red"
android:text="Hello, World!" />
```

Mint egy másik példa, mindig használjunk referenciákat, amikor string-eket használunk egy layout fájlban, lokalizálás miatt:

```
<EditText
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:textColor="@android:color/opaque_red"
android:text="@string/hello_world" />
```

5.9.2. Változó források és lokalizásás

Változó források és lokalizálás egy nagyon hasznos dolog.

Különösen, mikor tervezni kellene egy GUI-t, ami ugyanakkor jól illik a táj és potrté képernyő-orientációhoz – ami szinte lehetetlen.

Az alkalmazás ellátható különböző forrásokkal, az UI nyelvnek vagy a készülék hardver konfigurációjának megfelelően. Meg kell jegyezni, hogy habár tartalmazhat különböző string-eket, layout-ot, és minden más forrást, az SDK nem fedi fel a metódusokat, hogy specifikálhasd melyik változó forrás halmazt használod. Az Android kimutatja a hardver megfelelő beállításait és helyzetét, és helyesen tölti be őket. Csak a felhasználónak kell kiválasztani a változó nyelv beállításokat a készülék beállító paneljét használva. A változó források tartalmazásához párhuzamos könyvtárakat kell készíteni kötőjellel elválasztott minősítőket a könyvtár névhez csatolva, jelezve a konfigurációt, amit alkalmaz (nyelv, képernyő-orientáció, dpi, felbontás, …). Például, itt egy projekt az angol és német érték-források közötti eltérésekkel (csak string-ek):

```
MyApp/
res/
values-en/
strings.xml
values-de/
strings.xml
```

Az Android minősítők számos típusát támogatja, mindegyikre változatos értékekkel. Ezeket hozzá kell fűzni a forrás könyvtár név végéhez kötőjelekkel elválasztva. Minden könyvtár névhez több minősítőt lehet adni, de abban a sorrendben kell megjelenniük, ahogy itt listázva vannak. Például, egy könyvtár, ami rajzolható forrásokat tartalmaz egy teljesen specifikált konfigurációra, így nézhet ki:

```
MyApp/
res/
drawable-en-rUS-port-92dpi-finger-keyshidden-12key-dpad-480x320/
```

Még tipikusabb, ha csak specifikálsz néhány speciális konfiguráció beállítást, ami egy forrás számára van definiálva. Ki lehet dobni néhány értéket a teljes listából, amennyiben a maradék értékek még mindig ugyanabban a sorrendben vannak:

```
MyApp/
res/
drawable-en-rUS-finger/
drawable-port/
drawable-port-160dpi/
drawable-qwerty/
```

Az Android kiválogatja, hogy melyik változatos mögöttes forrás fájlok a legmegfelelőbbek a futási idő alatt, a jelenlegi készülék konfiguráció függvényében.

5.9.3. A varázslatos R.java

Egy projekt R.java –ja egy auto-generált fájl, ami indexeli az összes forrást a projektben. Ezt az osztályt a forráskódban használhatod, mint egy fajta rövidebb út a forrásra hivatkozáshoz, amit a projekt tartalmaz. Az ID-k kódkiegészítő tulajdonsága nagyon erős mint az Eclips-é mert gyorsan és interaktívan megtalálja a speciális referenciákat, amiket keresel. Továbbá fordítási biztonságot nyersz, hogy a forrás, amit használni akarsz, tényleg létezik. [5]

6 BanKdroid - Mobil banki alkalmazás Android platformon

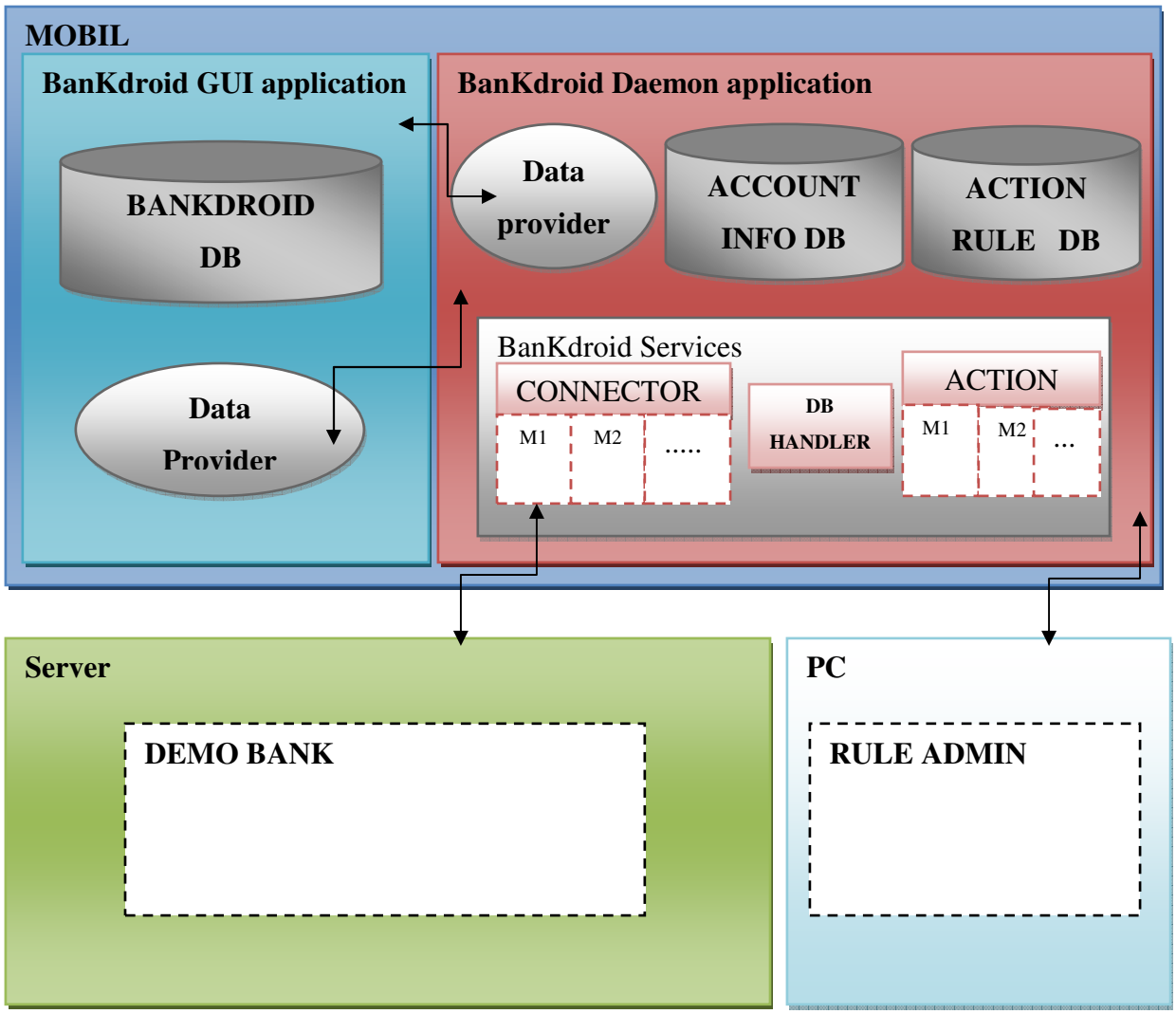
A célom egy olyan mobil banki kliens alkalmazás fejlesztése volt android platformon, ami a felhasználók számára információkat nyújt a számláik állapotáról értesítésként, amit saját maga adhat meg szabályok formájában (pl.: figyelmeztetést adjon mikor egy bizonyos számlán egy adott határ alá esik az egyenlege), illetve egy olyan felület elkészítése, ahol megnézheti számláinak és azokhoz kapcsolódó tranzakciók részleteit offline módban, amelyek egy titkosított adatbázisban tárolódnak.

Továbbá fontos szempontnak tartottam, hogy könnyen hozzáadható legyen egy másik bankhoz kapcsolódó megoldás, illetve könnyen hozzáadható legyen új szabályt kiértékelő megoldás a rendszer szerkezeti megbontása nélkül (pl.: az egyik számlánkon bizonyos összeg felé esik az egyenleg, akkor utaljon át adott összeget a megtakarítási számlára). Ezek mint szolgáltatási modulok helyezkednek el a rendszerben.

A következőkben ismertetni fogom ezek tényleges megoldását, ahol részletesen kitérek mind a tervezés, mind a megvalósítás részleteire. Ismertetem a megvalósított funkciókat, kisebb forráskód részletekkel bemutatom azoknak a problémáknak a megoldásait, amelyek gyakran előfordultak a kliens megvalósításakor.

A végén pedig bemutatom az IND Internet-banki rendszert és ahhoz a lehetséges kapcsolódási technológiákat a BanKdroidból.

6.1. A BanKdroid architektúrája



10. ábra: A BanKdroid architektúrája

Ahogy az ábrán is látható, a BanKdroid két alkalmazásból épül fel, amely két android projektből és egy java projektből tevődik össze.

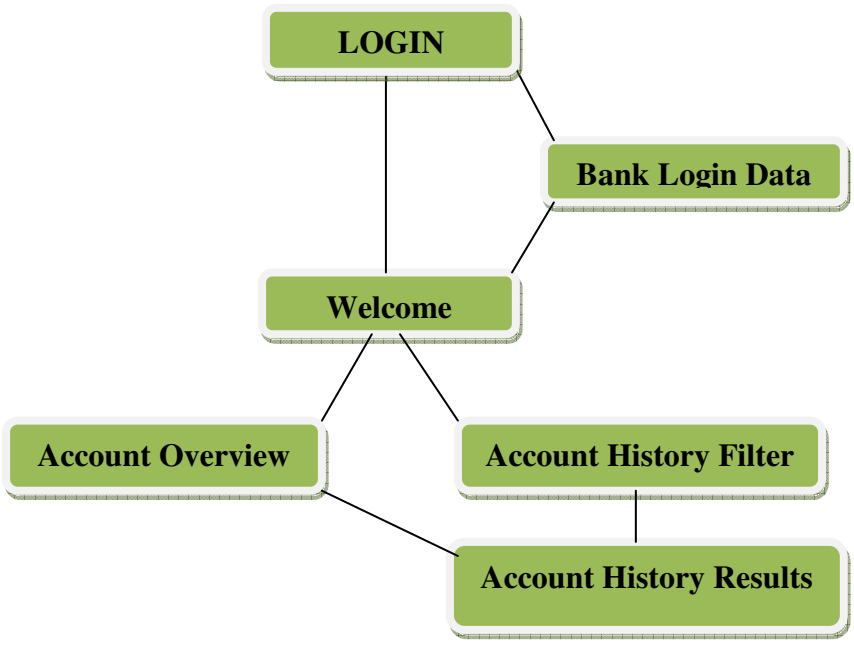
A BanKdroid Daemon-hoz tartozik a BanKdroidService java projekt, ebben találhatóak a szolgáltatási funkciók, mint a bankokhoz kapcsolódás, akciók kiértékelése és az adatbázis kezelésének megvalósítása.

6.2. BanKdroid GUI

A BanKdroid azon része, amelyen keresztül a felhasználó megnézheti a számláihoz tartozó információkat (egyenleg, számla státusza, …) és a számláihoz kapcsolódó tranzakciókat. Továbbá, itt adhatja meg a kapcsolódni kívánt bankhoz tartozó felhasználó nevét és jelszavát.

6.2.1. Megvalósított funkciók

- Login
- Welcome
- Bank Login Data
- Account Overview (Számla áttekintés)
- Account History (Számla történet)



11. ábra: Az alkalmazás képernyőfolyama

6.2.1.1. Login

A login képernyőn keresztül lehet belépni az alkalmazásba. Ha első alkalommal indítjuk el az alkalmazást, akkor ez a belépés egyben a jelszó megadását és mentését is jelenti.

6.2.1.2. Bank Login Data

A kapcsolódni kívánt bankhoz tartozó felhasználó nevet és jelszót lehet itt beállítani, amit majd a Daemon fog használni az adatok lekéréséhez az adott banktól.

6.2.1.3. Welcome

Sikeres belépéskor jelenik meg. (Itt különböző üzeneteket lehet megjeleníteni a felhasználó számára). Innen tudjuk a menün keresztül elérni az összes többi funkciót.

6.2.1.4. Account Overview

Itt jelennek meg a felhasználó számlaszámai és az ahhoz tartozó információk: számla név, elérhető egyenleg, könyvelt egyenleg, pénznem, számla státusza.

6.2.1.5. Account History

A számla történet funkció két felületből tevődik össze, a számla szűrésére vonatkozó adatok megadására szolgáló felületből és a szűrés eredményére vonatkozó tranzakciók listáját megjelenítő felületből. Szűrésként megadható egy számlaszám, a felhasználó számlaszámai közül, amin a tranzakciót hajtották végre és egy kezdő és egy végdátum, ami közé esik a tranzakció végrehajtásának dátuma.

Az eredmény képernyőn egy adott tranzakcióhoz megjelenik tranzakció dátuma, célszámlája, megjegyzés, összeg, egyenleg és a pénznem.

6.2.2. Egy felület általános felépítése

6.2.3. Technikai megoldások

Minden egyes képernyőn megjelenő funkciót megvalósító osztály egy *BanKdroidActivity* vagy a *BanKdroidListActivity* osztály leszármazottja, amelyek az android alaposztályaiból származnak az *Activity* és a *ListActivity*-ből.

Ezek nyújtanak a megvalósító osztály számára egy adatbázis kezeléséhez szolgáló *dbHandler* objektumot (amely biztosítja, hogy adatokat kérdezzünk, töröljünk, vegy beszúrjunk az alkalmazás adatbázisába), egy *request* nevű eljárást, amelynek hívásával a megvalósított osztályban lévő *doRequest* nevű eljárást hívja meg egy külön szálban, amely futása alatt egy folyamat dialógus ablakot jelenít meg a tevékenység jelzésére. A folyamat dialógusban megjelenített szöveget a request eljárás paramétereként kell megadni. Ha nem adjuk meg, akkor nem jelenik meg folyamat dialógus.

A *doRequest* eljárásban kell megadni az időigényes feladatokat pl.: a felület megjelenése után akarunk adatbázis műveletet végrehajtani. Ha a külön szálban lekérdezett eredményt meg

szeretnénk jeleníteni a felületen, akkor az android csak az activityt elindító szálban tudja ezt végrehajtani. Erre az android biztosít egy *Handler* nevű osztályt, amivel vissza tudunk térni az activyt elindító szálban. Megvalósító osztályban csak a *doSomethingWhenBackRequest* eljárást kell megvalósítani, mivel ez hívódik meg a külön szál lefutása után. Továbbá tartalmaz egy *toast* nevű objektumot, amellyel gyors információkat jeleníthetünk meg a felhasználó számára (pl.: rossz jelszó).

A BanKdroidActivity.java:

```
public abstract class BanKdroidActivity extends Activity implements
Runnable
{
   final private Handler handler = new Handler();
   private ProgressDialog progressDialog;
   protected DBHandler dbHandler;
   protected Toast toast;

   @Override
   protected void onCreate(Bundle savedInstanceState)
   {
         super.onCreate(savedInstanceState);
         dbHandler = new DBHandler(this, DBHandler.BANKDROID_DB);
         toast = Toast.makeText(this, "", Toast.LENGTH_LONG);
         create();
   }

   @Override
   protected void onStart()
   {
         super.onStart();
         start();
   }

   @Override
   protected void onPause()
   {
         super.onPause();
         dbHandler.close();
         dbHandler = null;
   }

   @Override
   protected void onResume()
   {
         super.onResume();
         if (dbHandler == null)
         {
               dbHandler = new DBHandler(this,
DBHandler.BANKDROID_DB);
         }
         resume();
   }

   protected abstract void create();

   protected abstract void start();

   protected abstract void resume();

   protected abstract void doRequest();

   protected abstract void doSomethingWhenBackRequest();
```

```
   protected void request()
   {
         Thread thread = new Thread(this);
         thread.start();
   }

   protected void request(int id)
   {
         progressDialog = ProgressDialog.show(this,
getResources().getText(R.string.progressdialogtitle),
getResources().getText(id), true, false);
         request();
   }

   public void run()
   {
         try
         {
               doRequest();
               progressDialog.dismiss();
               handler.post(requestFinish);
         }
         catch (Throwable t)
         {
               toast.setText(t.getMessage());
               toast.show();
         }
   }

   final Runnable requestFinish = new Runnable() {
         public void run()
         {
               doSomethingWhenBackRequest();
         }
   };
}
```

6.2.4. Adatbázis

A BanKdroid-ban db4o 7.4(http://db4o.com/) nevezetű adatbázist használtam és nem a beépített SQLite-ot, mivel az SQLite-ról a kutatásaim alapján megállapítható, hogy nem tudja titkosított formában tárolni az adatokat, ellenben a db4o igen. A db4o-nak megadható, hogy milyen titkosítást használjon, akár saját titkosítási algorítmust is. Én egy ingyenes letölthető XTEA (eXtended Tiny Encryption Algorythm) algoritmust használtam (http://en.wikipedia.org/wiki/XTEA).

A titkosítás megadása:

```
private Configuration dbConfig()
   {
         Configuration c = Db4o.newConfiguration();
         c.io(new XTeaEncryptionFileAdapter("BanKdroidDBHandler"));
         return c;
   }
```

A db4o egy nyílt forrású alkalmazás, ami elérhető Java-ban és .NET-ben is. Az is megtetszett, hogy egy objektumként tudja kezelni az adatbázis rekordokat, ahol az osztályváltozók lesznek a táblaoszlop nevei, és értékei a tábla egy record értéke. Viszont számos problémába ütköztem fejlesztésem során, mivel az android elég SQLite adatbázisra van kiélezve (Ez főleg a ContentProvider megvalósításnál jött elő).

Példa egy record beszúrására és lekérdezésére:

```
public void insertUser(UserTableRow user)
   {
         db().store(user);
   }

   public ObjectSet<UserTableRow> getUser(UserTableRow user)
   {
         return db().queryByExample(user);
   }
```

6.2.4.1. Kezelése

Adatbázis kezelésére a *DBHandler* osztály szolgál. Ez megvalósítja az adatbázis megnyitását, lezárását és az összes adatbázis műveletet, amit ez az alkalmazás használ függvények és eljárások formájában.

Az adatbázist megnyitó és bezáró függvény és eljárás:

```
   private ObjectContainer db()
   {
         synchronized (lock)
         {
               try
               {
                     if (oc == null || oc.ext().isClosed())
                           oc = Db4o.openFile(dbConfig(),
banKdroidDBFullPath());
                     return oc;
               }
               catch (Exception e)
               {
                     Log.e(DBHandler.class.getName(), e.toString(),
e);
               }
               return null;
         }
   }

   public void close()
   {
         if (oc != null)
               oc.close();
   }
```

Ezen alkalmazás adatbázisában csak egy USER nevű tábla van, amelyben tárolódik a BanKdroid alkalmazás belépéséhez szükséges jelszó és egy adott bank kapcsolódásához szükséges belépési azonosító és jelszó (oszlop nevek: BANKID, USERNAME, PASSWORD).

6.3. BanKdroid Daemon

A BanKdroid azon része, amely végzi az adatok szinkronizálását (számla információk, számla történetek) az adott bank és a Daemon adatbázisa között, értesítési szabályok kiértékelését és azok eredményeinek megjelenítését.

6.3.1. Technikai megoldások

A Daemon, mint ahogy a neve is utal rá, egy háttérben futó alkalmazás, amely a telefon indulásakor indul és bizonyos időközönként hajtja végre a szinkronizálást és a szinkronizálás eredményéből az értesítési szabályok kiértékelését és megjelenítését.

6.3.1.1. Daemon elindítása a háttérben

A telefon elindulásakor, amikor az android befejezte a boot-olási folyamatot, akkor egy BOOT_COMPLET eseményt szór szét, amit egy BroadcastReceiver típusú osztályban kezelhetünk le. Ez a DaemonServiceManager osztály.

A DaemonServiceManager osztály:

```
public class DaemonServiceManager extends BroadcastReceiver {

    public static final String ACTION =
"android.intent.action.BOOT_COMPLETED";
    @Override
    public void onReceive(Context context, Intent intent) {
          if( ACTION.equals(intent.getAction()))
          {
                ComponentName comp = new
ComponentName(context.getPackageName(),
DaemonService.class.getName());
                ComponentName service = context.startService(new
Intent().setComponent(comp));
                if (service == null)
                {
                      Log.e(DaemonServiceManager.class.getName(),
"Could not start service " + comp.toString());
                }
          }
          else
          {
                Log.e(DaemonServiceManager.class.getName(), "Received
unexpected intent " + intent.toString());
          }
    }
}
```

A DaemonServiceManager osztály beállítása az AndroidManifest.xml ben:

```
…
   <uses-permission
android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
…
    <application  android:label="@string/app_name">
…
        <receiver    android:name=".service.DaemonServiceManager"
                          android:enabled="true"
                          android:exported="false"
                          android:label="DaemonServiceManager">
            <intent-filter>
                <action
android:name="android.intent.action.BOOT_COMPLETED" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
         </receiver>
…
```

Ez hívja meg a *Service* osztályt, ami az androidban a háttérben futó folyamatok megvalósítására szolgál. Ez addig fut, amíg le nem állítjuk vagy ki nem kapcsoljuk a telefont. Ezt a szerepet az alkalmazásban a *DaemonService* nevű osztály tölti be. Ez az osztály valósítja meg a szinkronizálás, az értesítési szabályok kiértékelésének és megjelenítésének a meghívását bizonyos időközönként.

6.3.2. Könnyen bővíthetőség megvalósítása

Mint ahogy a bevezetőben is írtam, célom volt, hogy könnyen bővíthető legyen a rendszer, ha új bankhoz kapcsolódást és ezen bank szolgáltatásait kell megvalósítani, vagy új értesítési szabály kiértékelését a rendszer szerkezeti megbontása nélkül.

Erre a *ServiceLoader* nevű osztályt akartam használni, ami az 1.6 Java-ban jelent meg, viszont ezt az Android Java-ja nem tartalmazta, ezért ezt az osztályt kiemeltem és átírtam, hogy megfelelően működjön az alkalmazásban Android alatt. Ez lett az AndroidServiceLoader.

Az *AndroidServiceLoader* osztály segítségével egy adott bankra és egy értesítési szabályra vonatkozó szolgáltatási konfigurációkat tudok betölteni, amin keresztül könnyen kezelhető az adott szolgáltatás. Minden szolgáltatási konfiguráció a szolgáltatást megvalósítását leíró interface típusú kell, hogy legyen.

Például a különböző bankok szolgáltatásának leírója:

```
public interface BankServicePluginConfiguration
{
   public Bank getBank();

   public String getServicePackage();

   public String getServicePrefix();

   public BankService getBankService( Class<?> serviceClass ) throws
BankServiceException;
}
```

Minden egyes konkrét banki szolgáltatás konfigurációja ezt az interface-t kell, hogy implementálja.

Az *AndroidServiceLoader load* függvényével könnyen felolvashatók ezek a szolgáltatást leíró konfigurációs osztályok. A felovasáshoz létre kell hozni a *com.bankdroid.services* csomagban egy *com.bankdroid.connector.bank.BankServicePluginConfiguration* nevű UTF-8 kódolású fájlt, amiben fel kell sorolni az adott bank konfigurációs osztályát, teljes útvonallal.

A *com.bankdroid.services* csomag tartalma:

A *com.bankdroid.connector.bank.BankServicePluginConfiguration* tartalma:

```
com.bankdroid.connector.dummy.DummyPluginConfiguration
```

További bank konfigurációs osztályt is itt kell megadni.

Ezek után csak az *AndroidServiceLoader load* függvényét kell meghívni, aminek paraméterében a *BankServicePligunConfiguration.class-t* kell megadni és az *AndroidServiceLoader* a *com.bankdroid.services* csomgban megkeresi a *com.bankdroid.connector.bank.BankServicePluginConfiguration* fájlt és betölti a benne lévő osztály nevek alapján az adott osztályokat.

A *load* függvény visszatérési értéke ezen osztályok listája lesz.

Ezek után könnyen meghívhatunk egy adott bankhoz tartozó szolgáltatás megvalósítását, például a bankba való belépést.

A bankba való belépés meghívása:

```
LoginService loginService = BankServiceFactory.getBankService(bank,
LoginService.class, context);
```

Különböző bankok azonos szolgáltatásainak egy közös interfészt kell megvalósítaniuk (például a belépésnél, a *LoginService* interface-t), amely a *BankSercice* interface kiterjesztése. A *BankService* interface egy *execute* nevű abstract eljárást tartalmaz. Ennek segítségével egységesen történhet a különböző bankok azonos szolgáltatásainak a meghívása és eredményeinek feldolgozása.

Hasonló a működés az értesítési szabályok kiértékelésének bővíthetőségénél is.

6.3.3. Adatbázis

Hasonlóan a GUI alkalmazáshoz, tartalmaz egy DBHandler osztályt, amely megvalósítja az adatbázis megnyitását, lezárását és az összes adatbázis műveletet, amit ez az alkalmazás használ függvények és eljárások formájában.

Ez az alkalmazás két adatbázis sémát tartalmaz, a BANKDROID_TRANSACTIONINFO.DB és a BANKDROID_RULE.DB-t, melyek szintén titkosított formában tárolják az adatokat.

A BANKDROID_TRANSACTIONINFO.DB két táblát tartalmaz, az ACCOUNTOVERVIEW táblát, amely tartalmazza az accountra vonatkozó információkat és az ACCOUNTHISTORY-t, amely tartalmazza a számlákhoz kapcsolódó tranzakció történet részleteket.

A BANKDROID_RULE.DB egy táblát tartalmaz, amely most csak a BALANCERULE tábla, amely tartalmazza az egyenleg értesítésre vonatkozó szabályokat. A tábla a következő oszlopokat tartalmazza: BANKID – melyik bankra vonatkozik a szabály, ACCOUNTNUMBER - melyik számlaszámra, LOGICOPERATOR - milyen összehasonlítás legyen, COMPAREAMOUNT -

összehasonlítási összeg, MESSAGE - milyen szöveg jelenjen meg ezen szabály teljesülése esetén, PRIORITY - mennyire fontos.

6.3.4. Alkalmazások közötti adatcsere

Fontos megemlíteni, hogy az Androidban az adatbázisok egy adott alkalmazáshoz tartozhatnak és más alkamazás közvetlenül nem férhet hozzá. Az Andorid az alkalmazások közötti adatcserére egy ContentProvider nevezetű osztályt nyújt, amely osztály fölüldefiniálásával valósíthatjuk meg az adott alkalmazás adatainak megosztását más alkalmazások számára.

Az AndroidManifest.xml fájlban meg kell adni, hogy melyik ez a ContentProvider osztály és milyen authoritin keresztül férhetünk hozzá.

```
<provider android:name=".provider.DataProvider"

android:authorities="com.bankdroid.daemon.provider.dataprovider">
        </provider>
```

Számlákhoz kapcsolódó adatok lekérdezését a Daemon alkalmazástól a következő módon tehetjük meg:

```
accounts =
this.getContentResolver().query(Uri.parse("content://com.bankdroid.d
aemon.provider.dataprovider/accountoverview"), null, null, new
String[] { bankId }, null);
```

Ez fontos volt a BanKdroid alkalmazás számára, mivel a GUI-nak a számla információkat és a számlákhoz tartozó tranzakciókat a Daemontól kellet elkérnie, a felhasználó adatokat pedig a Daemon kellett, hogy elkérje a GUI-tól.

6.4. IND Internet-banki rendszerének rövid bemutatása

Az IND Secure Transaction Server 2005 (IND STS) platform az IND Banking Front-Office megoldáshoz, amely Java EE technológiára épül, és az elektronikus pénzügyi szektor kiszolgálására készült. Az STS költséghatékony tranzakció kezelést biztosít a bankok számára. A termék fő célja, hogy szabályos működést biztosítson a bankok ügyfél oldali front-office alkalmazásainak, amely lehet IND vagy más szállító terméke. Az STS a pénzügyi szolgáltatásokat nyújtó ipar kiszolgálására lett tervezve, elsősorban kis és közép méretű bankok

számára. Ezért számos olyan funkciót tartalmaz, amely növeli a pénzügyi szolgáltatók szolgáltatásainak értékét, és egyben a felhasználók igényeit is kielégíti. Mindezek mellett az STS nem csak az elektronikus banki ügyintézés csatornáinak (internet, fióki elérés, ivr, call center, wap, sms, fax) egy rendszerrel történő kiszolgálásának eszköze, hanem egy fejlesztői keretrendszer is. A fejlesztés Java és az XML technológiákkal történhet, amelyek maguk után vonják a platformfüggetlenséget. A fejlesztés során XML-ben történik a rendszer programozása: adattípusok leírása, tranzakciók leírása, háttérrendszerek elérése. Ezek az XML-ben definiált funkcionalitások, adatszerkezetek újrafelhasználhatóak.

6.4.1. STS komponensei

Az IND Secure Transaction Server a Java EE alkalmazás szerveren és technológián alapszik, továbbá a komponensek három fő csoportjából áll:

1. A *Channel Framework* biztosítja a felhasználóval való kommunikációt a különböző csatornákon. A mobil kliensalkalmazás a rendszerrel történő kommunikáció során ezzel a komponenssel van közvetlen kapcsolatban. Ez a komponens fogadja a kéréseket, továbbítja azokat a tranzakció feldolgozó modulhoz, valamint küldi vissza a választ a mobil kliensnek.
2. A *Transaction Handling* komponensek gondoskodnak a Channel Framework-től érkezett tranzakciók biztonságos és szabályszerű feldolgozásáról.
3. A *Backend Connector Framework* továbbítja a feldolgozott tranzakciókat a backend rendszerek felé és fogadja azok nyugtázási üzeneteit. Backend rendszereken értjük a bankok saját tranzakció- és, számlakezelő információs rendszerét.

Számos bank használja a Secure Transaction Server-t, hogy megvalósítsa a saját SOA (Service Oriented Architecture) koncepcióját. Az alkalmazások, amelyek az STS-en futnak, közzé tudják tenni szolgáltatásaikat és képesek más alkalmazások szolgáltatásainak igénybevételére. Az STS middleware szerepet is játszik a bankügyi front-office és back-office alkalmazások között, azért mert a backend rendszerek funkcionalitásai az STS-en keresztül érhetők el az ügyfelek számára az elektronikus és off-line csatornákon. Az STS fogadja, feldolgozza, és továbbítja a különböző csatornákon érkező felhasználói kéréseket.

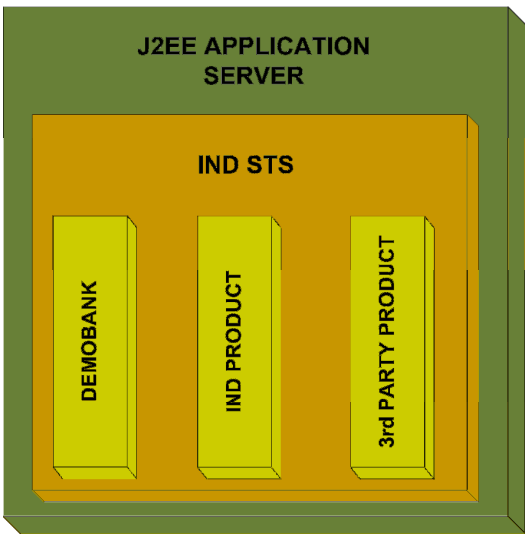
Az STS rendszerkövetelményei egy Java EE 1.3-nak megfelelő alkalmazás szerver és egy ahhoz megfelelő adatbázis rendszer.

Az STS az alábbi termékek támogatását biztosítja:

- *Alkalmazás szerverek:* BEA WebLogic Server, IBM WebSphere Application Server, SUN Java Systems Application Server, Oracle Java Application Server
- *Adatbázis rendszerek:* IBM DB/2, Microsoft SQL Server, ORACLE DBMS

6.4.2. A Demobank alkalmazás

Mint fentebb említettem, az STS banki alkalmazásokat futtat, így szükség volt egy olyan rendszerre, amely megvalósítja a szükséges banki funkciókat. Az IND STS 2005 terméke magában foglal egy úgynevezett Demobank modult, amely implementál néhány alap banki funkciót. Ilyen például az átutalás, számlatörténet, számlaáttekintés stb. Mint nevéből is kiderül, ez egy demo banki alkalmazás, amely nem hasonlítható össze egy igazi Internet-bankkal, amely az elérhető funkciók jóval szélesebb skáláját teszi elérhetővé. Felhasználását tekintve legtöbbször a cégen belüli oktatásokra, valamint az eladással foglalkozó alkalmazottak prezentációihoz használatos. Ami tehát fontos, hogy fusson egy alkalmazás az STS-en, amely funkcióit - mint már említettem - az STS elérhetővé teszi a különböző csatornákon.



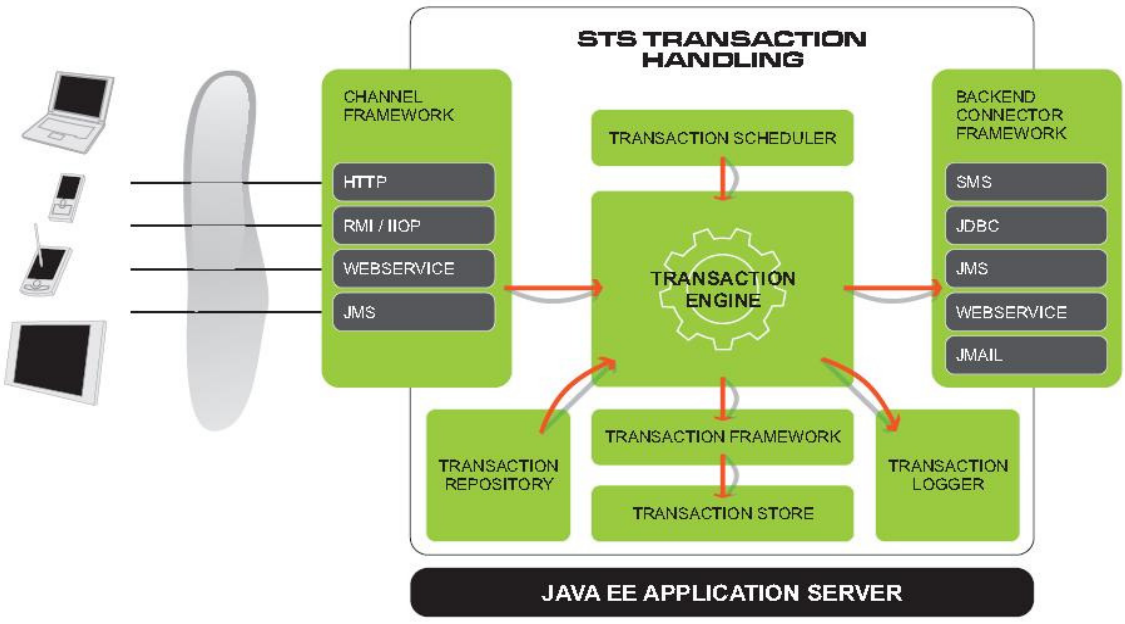
12. ábra: Az internetbanki rendszer felépítése

6.4.3. Lehetséges kapcsolódási felületek a mobil banki kliens számára az IND Internet-banki rendszerében

Az STS Channel Framework komponense jelenleg négy protokollon képes tranzakciós műveletek fogadására, amelyeken keresztül minden, az STS-re telepített alkalmazás szolgáltatását elérhetővé teszi. Ez a négy csatorna a

- HTTP
- RMI/IIOP
- Webservice
- JMS csatorna

A felsorolt protokollok mindegyikének megvannak a maga előnyei és hátrányai.



13. ábra: STS komponensei és kacsolódási felületei

7 Összefoglalás

A dolgozat írása során elsőként a mobil eszközök kialakulását, típusait, elterjedésüket ismertettem és bemutattam azok előnyeit és hátrányait.

Majd a mobil eszközökön gyakran használt szolgáltatások közül a mobil bankolást fejtettem ki. Bemutattam a leggyakrabban használt mobil banki ügyintézéseket és ismertettem a lehetséges mobil bankolásra alkalmas technológiákat.

A technológiák közül a mobil kliens alkalmazást (Standalone Mobile Application Client) választottam ki a BanKdroid nevű mobil banki kliens alkalmazás megvalósítására Android platform alatt.

Ismertettem az Android-ot, amely nem csak az általános mobil banki funkciók implementálására nyújt nagyszerű eszközöket, hanem számos új lehetőségeket is rejt. Habár az Android platform és technológia még nagyon fiatal, számos olyan megoldás rejlik benne, amellyel meglepően egyszerűvé tette a fejlesztést, sejtetve a rendszer megvalósítása előtti alapos tervezési munkát. Ilyen eszköz volt az Eclipse fejlesztőkörnyezethez készített plugin, amely az Android Development Tools (ADT) nevet kapta. Segítségével az Android fejlesztőeszközei az Eclipse-en belülről használhatóak, felgyorsítva és automatizálva ezzel a fejlesztés folyamatát. Az egyik ilyen eszköz az Android emulátor, amely grafikus felületén kívül számos, a fejlesztést segítő funkciót is tartalmaz.

Végül a BanKdroid alkalmazás megvalósítás fontosabb lépéseit mutattam be. Olyan BankDroid nevű alkalmazást valósítottam meg, ami a felhasználók számára információkat nyújt a számláik állapotáról értesítésként, amit saját maga adhat meg szabályok formájában, illetve egy olyan felület elkészítése, ahol megnézheti számláinak és azokhoz kapcsolódó tranzakciók részleteit offline módban, amelyek egy titkosított adatbázisban tárolódnak.

Továbbá az alkalmazáshoz könnyen hozzáadható egy másik bankhoz kapcsolódó megoldás, illetve könnyen hozzáadható új szabályt kiértékelő megoldás a rendszer szerkezeti megbontása nélkül.

8 Köszönetnyilvánítás

Szeretnék köszönetet mondani mindenkinek, aki közvetlen vagy közvetett módon segítette a dolgozat elkészülését. Köszönöm az IND Kft.-nek a diplomatémát; külső konzulensemnek Gyenes Gábornak a hasznos tanácsait és hogy felhívta figyelmem az Android technológiában rejlő lehetőségekre.

Köszönettel tartozom belső konzulensemnek Dr. Fazekas Gábor egyetemi docensnek, a munka menetének felügyeletéért.

Végül szeretném megköszönni feleségemnek a segítségét és türelmét.

9 Irodalomjegyzék

[1] (2007) Mobiltelefonok története

http://en.wikipedia.org/wiki/History_of_mobile_phones

[2] (2007) International Telecommunication Union

http://www.itu.int/ITU-D/ict/statistics/ict/index.html

[3] (2008 szeptember) Nemzeti Hírközlési Szolgálat

http://www.nhh.hu/index.php?id=hir&cid=6031

[4] Mobil bankolás

http://www.tutorial-reports.com/mobile/mobile-banking/services.php

[5] (2008) Android

http://www.android.com/

[6] (2008) db4o adatbázis

http://db4o.com/

Melléklet

1. melléklet: BanKdroid

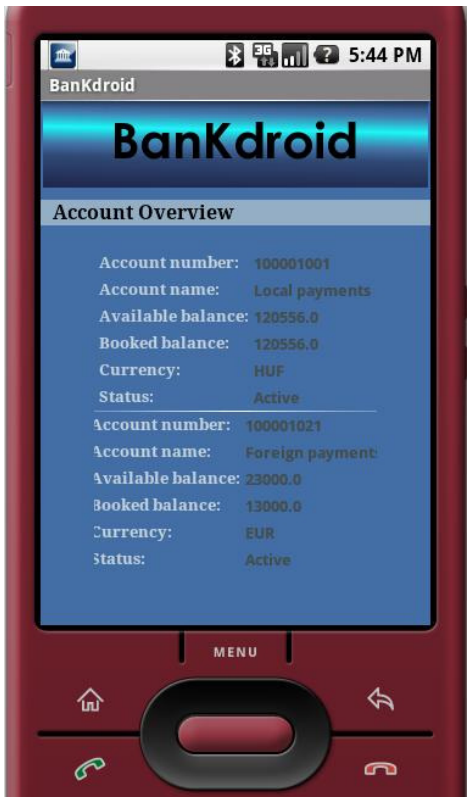

5:44 PM
BanKdroid
BanKdroid
Login
Password:
Save
Your first visit BanKdroid application.
Please set password!
MENU


5:44 PM
BanKdroid
BanKdroid
Bank Login Data
Login Id:
Password:
Confirm password:
Save
MENU


5:44 PM
BanKdroid
BanKdroid
Bank Login Data
Login Id:
dummy
Password:
Confirm password:
Some fields are empty or password and
confirm password not same.
MENU


5:44 PM
BanKdroid
BanKdroid
Welcome
Welcome to the BanKdroid!
Account Overview
Account History
Login Data
Logout
MENU


5:44 PM
BanKdroid
BanKdroid
Account Overview
Working ...
Load accounts.
MENU


5:44 PM
BanKdroid
BanKdroid
Account Overview
Account number: 100001001
Account name: Local payments
Available balance: 120556.0
Booked balance: 120556.0
Currency: HUF
Status: Active
Account number: 100001021
Account name: Foreign payment
Available balance: 23000.0
Booked balance: 13000.0
Currency: EUR
Status: Active
MENU


5:44 PM
BanKdroid
BanKdroid
Account History
Transaction date: 11-24-2008
Account number: 100001021
Partner account number: 100001021
Amount: 517.0
Balance: 50152.0
Currency: HUF
Description: Test transaction 0
Transaction date: 11-26-2008
Account number: 100001021
Partner account number: 100001022
Amount: 34.0
Balance: 50186.0
Currency: HUF
MENU


5:44 PM
BanKdroid
BanKdroid
Account History
Account number: 100001001
From date: 11-24-2008
To date: 12-24-2008
09
09
Search
MENU


5:44 PM
BanKdroid
BanKdroid
Account History
100001001
100001021
MENU


5:44 PM
BanKdroid
BanKdroid
Mon, November 24, 2008
Nov
24
2008
Set
Cancel
MENU


December 24, 2008
5:44 PM
Android
Clear notifications
Notifications
Dummy Bank SA - Balance not...
Your balance has fallen under 1300... 6:04 PM
MENU